Compaction Algorithms for Non-Convex Polygons and Their Applications

Compaction Algorithms for Non-Convex Polygons and Their Applications

A thesis presented

by

Zhenyu Li

to

The Division of Applied Sciences

in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

in the subject of

Computer Science

Harvard University

Cambridge, Massachusetts

May, 1994

Contents

List of Figures

Abstract

Given a two-dimensional, non-overlapping layout of convex and non-convex polygons, *compaction* refers to a simultaneous motion of the polygons that generates a more densely packed layout. In industrial two-dimensional packing applications, compaction can improve the material utilization of already tightly packed layouts. Efficient algorithms for compacting a layout of non-convex polygons are not previously known.

This dissertation offers the first systematic study of compaction of non-convex polygons. We start by formalizing the compaction problem as that of planning a motion that minimizes some linear objective function of the positions. Based on this formalization, we study the complexity of compaction and show it to be PSPACE-hard.

The major contribution of this dissertation is a position-based optimization model that allows us to calculate directly new polygon positions that constitute a locally optimum solution of the objective via linear programming. This model yields the first practically efficient algorithm for translational compaction–compaction in which the polygons can only translate. This compaction algorithm runs in almost real time and improves the material utilization of production quality human-generated layouts from the apparel industry.

Several algorithms are derived directly from the position-based optimization model to solve related problems arising from manual or automatic layout generation. In particular, the model yields an algorithm for separating overlapping polygons using a minimal amount of motion. This separation algorithm together with a database of human-generated markers can automatically generate markers that approach human performance.

Additionally, we provide several extensions to the position-based optimization model. These extensions enables the model to handle small rotations, to offer flexible control of the distances between polygons and to find optimal solution to two-dimensional packing of non-convex polygons.

This dissertation also includes a compaction algorithm based on existing physical simulation approaches. Although our experimental results showed that it is not practical for compacting tightly packed layouts, this algorithm is of interest because it shows that the simulation can speed up significantly if we use geometrical constraints to replace physical constraints. It also reveals the inherent limitations of physical simulation algorithms in compacting tightly packed layouts.

Most of the algorithms presented in this dissertation have been implemented on a SUN SparcStation^{TM} and have been included in a software package licensed to a CAD company.

Acknowledgement

This thesis is the product of my work on the automated marker making project at Harvard University. I would like to thank the Alfred P. Sloan Foundation and the Textile/Clothing Technology Corporation for funding the project.

I am much indebted to my thesis advisor Professor Victor Milenkovic of Harvard University for the huge amount of effort he spent in guiding my research. He is always there to share his quick wit, to clarify and enhance my thinking and to spark new ideas. Without his timely guidance, this thesis would not have been completed. His engineering savvy and active leadership in the marker making project is indispensable for the successful implementation of the compaction algorithms presented in this thesis. This whole thesis can be considered joint work with him. I thank him for being a guru in almost every aspect.

Many thanks to Professor Fred Abernathy – the principal investigator of a project of which the marker making project is a part. I thank him for his encouragement, for offering his insight on textile and apparel industry and for enlightening and enjoyable conversations.

Many thanks to Professor Harry Lewis and Professor Les Valiant for serving in my program committee and for their interests in my thesis work.

Special thanks to Karen Daniels for being a wonderful officemate and a great "teammate" on the marker making project. We have shared the learning process of several courses and had fruitful cooperations on the marker making project. Her careful reading and criticism of my writings have been extremely helpful.

Thanks to Professor Leonidias Guibas for his illuminating computational geometry class at MIT; to Dr. Richard Szeleski of DEC Cambridge Research Laboratory for helpful discussions and for drawing our attention to the physically based simulation method; to Murray Daniels (Karen's husband) of MITRE Corporation for providing user interface software and for his generous help in customizing the software; to Binhai Zhu of McGill University for interesting discussions about computational geometry and a joint work.

I would also like to thank the undergraduate students who have worked on geometric algorithms under the supervision of Professor Milenkovic; most of them have made direct contributions to the marker making project: Rajarshi Bhattacharyya, Jackie Chang, Shivashish Chatterjee, Sanjoy Dasgupta, Jacqueline Huang, Matt LaMantia, Sanjay Madan, Kirat Singh, Venkatesh Reddy, and Lee Wexler. I would especially mention Sanjay Madan. He started working on the marker project in June 1993 and worked directly on the compaction software. He helped to reorganize the compaction software and to implement new

compaction functions.

I thank my fellow graduate students at Aiken Computation Laboratory for making it an interesting place to work. Thanks also to Carol Harlow and Baiba Menke for providing administrative assistance; to the computer system support staffs at Aiken for managing our project machines and for answering system related questions.

Lastly, I thank my wife for her understanding, support and love.

Chapter 1

Introduction

1.1 Automated Marker Making

1.1.1 Cutting and Packing Problems

In the past thirty years, a class of problems, categorized as *Cutting and Packing* problems [DD92] [Dyc89] [SP92b], have been topics of extensive study in both operations research and computer science because of their extremely wide application areas and their great theoretical challenge. Some of the well-known packing problems in the class include:

One-dimensional Bin Packing The problem of putting variable sized items (the sizes are represented as integers) into bins of the same capacity and minimizing the number of bins used, subject to the condition that the total size in each bin does not exceed the capacity [ECL91].

Cutting Stock An extension of the one-dimensional bin packing problem. Instead of having the same capacity, the bins have k different capacities, where k is fixed. There are unlimited supplies of bins for each capacity. Bins of the same capacity are assigned the same cost. The objective is to minimize the total cost while packing the items into these different sized bins [GG61] [GG63]. This problem is also called *variable sized bin packing* when the cost of a bin is proportional to its capacity [Mur87].

Two-dimensional Bin Packing The problem of packing a set of rectangles of different height and width into rectangular bins such that the rectangles do not overlap with each other. The bins are of the same width and height. Usually, there are additional restrictions on the orientation of the rectangles, such as the edges of a rectangle must

be parallel to those of the bin that contains it and no rotation of a rectangle is allowed. The objective is to use the minimal number of bins to pack all the rectangles [CMD82]. A variation of the problem is called *strip packing*: instead of using many bins of the same size, a bin of fixed width and unlimited length is used and the objective is to pack all the rectangles using minimum length [BECR80] [ECS90].

Two-dimensional Packing Generalizations of two-dimensional bin packing in which the objects being packed can be non-rectangular and the restrictions on orientation can also be lifted.

It is well-known that the one-dimensional bin packing problem is NP-complete [GJ79]. The NP-hardness of the other packing problems can be shown by a reduction to the one-dimensional bin packing problem. The two-dimensional packing problems in which objects are not allowed to rotate can be shown to be in NP (and therefore NP-complete) by an argument similar to the one used in [MDL91]. In spite of the extensive research efforts devoted to these problems, most of the problems still lack solutions that are satisfactory in practice – a fact that demonstrates the inherent difficulty of the problems.

Another interesting Cutting and Packing problem is that of packing the maximum number of copies of a single item. Typically, the material and the item are both rectangles, and only one orientation is allowed. There are variations involving non-rectangular material or item and multiple orientations [Gar79]. Closely related to this problem is the *covering problem*: what is the minimum number of copies needed to cover the sheet of material [CKSS81]? The *tiling* problem [G$\ddot{\text{o}}$79] is a combination of packing and covering: cover the whole plane with non-overlapping copies of a single item (or a few items).

1.1.2 Two-dimensional Packing and Marker Making

Problems of packing non-rectangular polygonal objects onto a rectangular strip of material are of central importance to the sheet metal, ship building, furniture, leather and textile industries. These problems are also known as *polygon nesting* or *layout* problems. The non-rectangular polygonal objects to be packed are also referred to as *irregular shapes* in the operations research literature. The objects represent pieces to be cut from a sheet of material. The output produced by a two-dimensional packing algorithm is called a *layout*.

In apparel manufacturing, the process of strip packing a set of polygons, which correspond to garment pieces, on a roll of cloth of fixed width but unlimited length is called *marker making*. The layout generated by the process is called a *marker*. In practice, a

marker is not the optimal strip packing but one of near minimum length. The shapes of garment pieces are mostly non-rectangular as depicted in Figure 1.1.

The material utilization, or the *efficiency*, of a marker is the ratio of the area occupied by the garment pieces to the area of the strip of cloth. A source in the textile industry estimates that about 90 percent of the cost in apparel manufacturing is due to the cost of material. Therefore, the efficiency is considered the most important factor in judging the quality of a marker.

Currently, marker making is done by experienced human marker makers with the assistance of interactive CAD systems. One of the easiest types of markers to generate is the pants marker. But even for this type of garment, a human marker maker needs about a half year of training before he/she can generate markers of acceptable efficiency. In about 45 minutes, an experienced human marker maker can generate a pants marker with efficiency of around 90 percent , which is estimated to be within 1 percent of the optimal. The marker shown in Figure 1.1 is generated by a human marker maker. Current automated marker making systems fall short of human performance by 5 to 10 percent in marker efficiency for pants markers.

1.1.3 Project Background

The results presented in this thesis are developed from the automated marker making project supervised by Professor V. Milenkovic at Harvard University. The goal of the project is to match human performance on pants markers.

Because of the difficulty of the general two-dimensional strip packing problem embodied in automated marker making, it is necessary to first concentrate on a relatively simple domain such as pants. Pants markers are chosen as the target because their pieces can be easily classified by size and the art of making high efficiency pants markers has been mastered by experienced human marker makers whose knowledge can provide valuable heuristics on algorithm design. It is hoped that the techniques developed in this project can be carried over to other domains such as jacket markers or blouse markers and other industries such as sheet metal and ship building.

In pants markers, there is a clear distinction between large pieces which are called *panels* and small pieces which are called *trim pieces*. As can be seen from Figure 1.1, the panels are the front and back pieces and the trim pieces are waist bands, packet facings, belt-loops, etc. From observing the marker making procedure of the best human marker makers, we

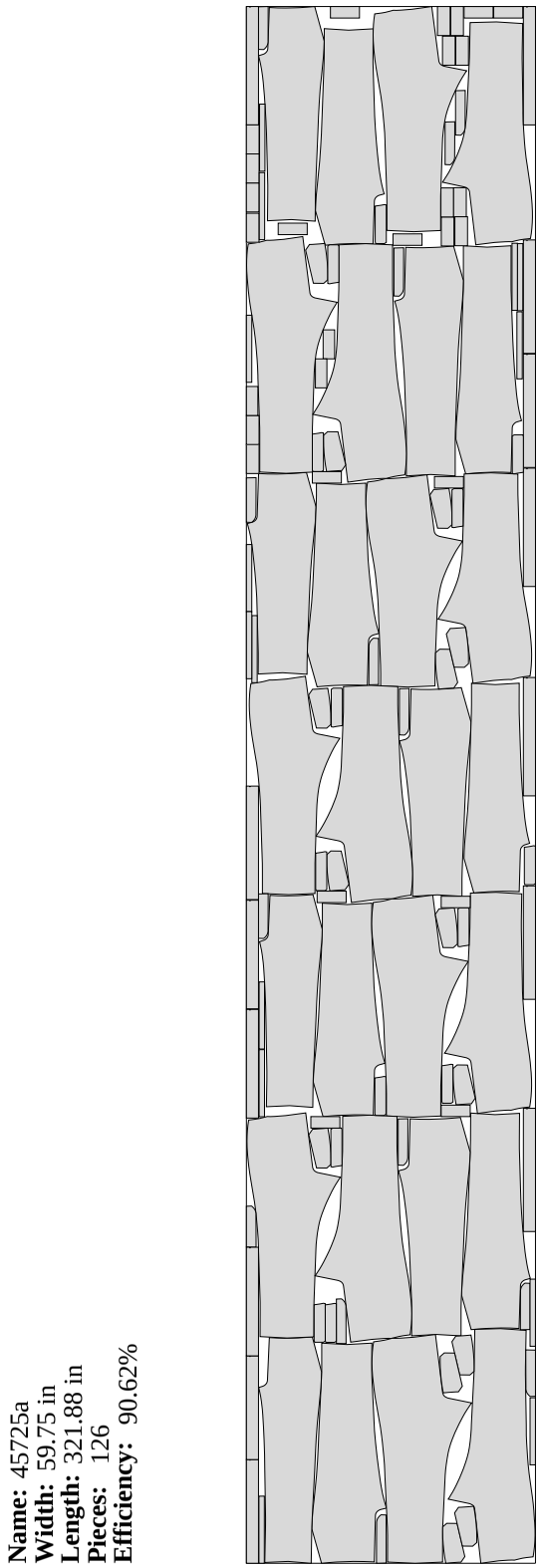


Figure 1.1: A human generated pants marker in apparel manufacturing.

discovered that the task of making pants markers is clearly divided into two steps: first place all the large panels and then place all the trim pieces in the "gaps" – the unoccupied areas left over between the large panels or between a large panel and the boundaries of the marker. Consequently, our project decomposes the placement task into two parts: the placement of large panels and the placement of the trim pieces.

In addition to panel and trim placement, a third part of our project focuses on solving a different problem which we call the *compaction* problem. Compaction takes a human or machine generated layout as input and generates a more efficient layout by simultaneous motion of the polygons. Our goal is to find practical algorithms for compaction. The results presented in this thesis are obtained from my work on the compaction part of the project.

1.1.4 Problem Representation for Marker Making

In marker making, the garment pieces are represented by polygons which are digitized from the garment patterns drawn by human pattern designers. In the rest of the thesis, we will use the terms *piece* and *polygon* interchangeably. The vertices of a polygon are the sample points on the boundary of a pattern, and each pair of vertices is connected by an edge (*i.e.* a straight line segment). There are more vertices and shorter edges in regions of higher curvature, and fewer vertices and longer edges in regions of lower curvature. In Figure 1.2, we identify the vertices in the upper left part of a pattern piece (piece 0) with small dots. There is a local coordinate system attached to each piece. The origin of the local coordinate system is usually fixed at the center of the *bounding box* of the piece: the smallest rectangle containing the piece with sides parallel to the coordinate axes. The coordinates of the vertices are specified in the local coordinate system. The position of piece in the marker is given with respect to a global coordinate system. The origin of the global coordinate system corresponds to the lower left corner of the rectangular sheet of cloth on which the pieces should be laid. The placement of the pieces is restricted to a horizontal rectangular strip in the first quadrant of the global coordinate system. The strip which represents a roll of cloth will be called the *marker region*

The marker region is bounded from the left by the y axis and is open on the right. After the panel placement phase, the uncovered regions remaining in the marker region are broken into a set of connected components [DM94]. We use the term *gaps* to refer to these components in trim placement and compaction. Gaps are represented as polygons.

The rules of marker making sometimes permit a piece to change its orientation by

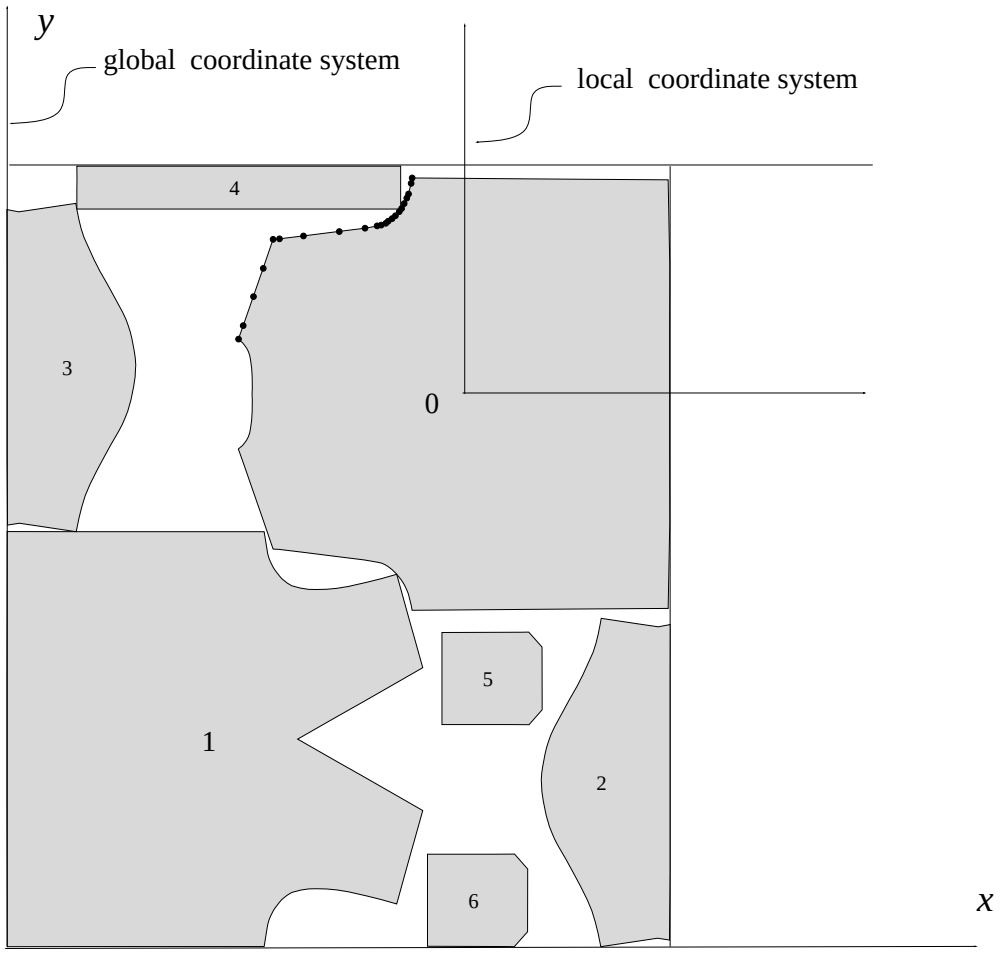


Figure 1.2: Marker coordinate systems and points on the boundary of a piece.

rotating and/or flipping in the local coordinate system. A basic rotation amount is specified for each piece. The amount is usually 45, 90 or 180 degrees. A piece can be rotated by a multiple of its basic rotation amount about its local origin. In addition to multiples of its basic rotation amount, a piece can also rotate by some small amount called *tilt* which is usually less than 3 degrees but can be as large as 7 or 8 degrees depending on the individual apparel manufacturer. In the sample production markers we have from a large pants manufacturer, more than 80 percent of the pants markers contain tilted pieces. *Flipping* of a piece is done with respect to the local x axis.

1.2 Compaction

1.2.1 Description and Examples

The central topic of this thesis is *compaction*. Informally, compaction can be thought of as the action of squeezing out the extra free spaces in an existing layout to produce a shorter layout and hence a tighter packing of the polygons. Suppose we view the pieces as rigid bodies and imagine that we could apply forces on them. Then we can make a tighter packing by pushing the right boundary of a marker to the left to shorten its length.

The analogy of compaction using rigid bodies and forces naturally leads to the formulation of compaction as planning a continuous motion for the polygons to minimize an objective "energy" function. Usually, the objective function is the area of the strip of material used by the layout which is equivalent to the length of the strip since its width is fixed. In this formulation, flipping a polygon or rotating it by its basic rotation amount (usually 90 or 180 degrees) are not allowed since they are not continuous motions. On the other hand, continuous motions such as translation and tilting a piece within its tilt limits are permitted.

Figure 1.3 shows an example of compaction. A human generated jacket marker is shown on the left. The compacted marker is shown on the right. The marker efficiency has been improved by more than 3 percent after compaction. In this example, polygons are allowed to translate and tilt.

1.2.2 Motivation

A marker acts as a cutting plan for a human or robotic cloth cutter. Usually, many layers are cut at the same time; also, the same marker may be used many times as a cutting plan. It follows that even a small percentage increase in efficiency for a marker results in a large savings in material. For pants manufacturing, each time a marker is used, sixty layers of cloth are cut at the same time, and a 1 percent increase in efficiency represents a savings of about $25. A representative of a large pants manufacturer estimates that a 0.1 percent average improvement in efficiency will save his company about two million dollars per year.

Compaction proves to be an extremely hard task for humans. Production quality markers are packed very tightly: every piece is touching its surrounding pieces, so there is no room for moving only one piece. Thus, moving one piece at a time can not shorten the marker length. We can verify this fact by the following strategy. First, order the pieces

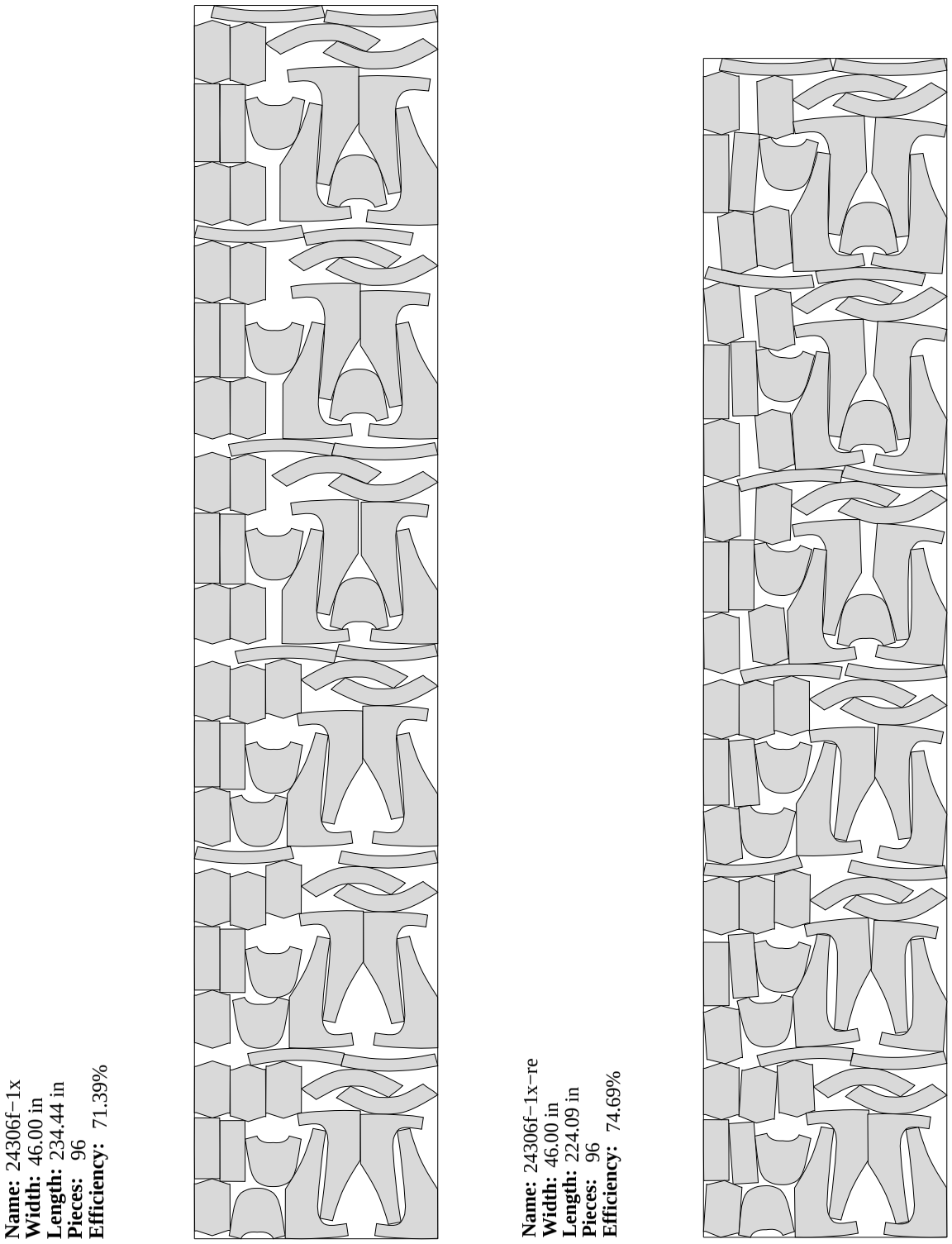


Figure 1.3: Another leftward compaction example. Left: A human generated jacket marker. Right: The marker after leftward compaction.

from left to right according to the left boundary of the bounding box of each piece. Next, visit the pieces sequentially according this order, and move each piece as far to the left as possible without overlapping other pieces.[1] Most of the production markers we tested using this strategy did not show improvements in efficiency.

Therefore, the ability to move many pieces at the same time is essential to compaction. However, it is beyond the capability of human marker makers to handle the simultaneous motion of many, possibly hundreds of pieces[2]. This gives rise to the demand for efficient computer algorithms for compaction. In addition to the direct benefit of cost savings, compaction algorithms can also facilitate manual or automated marker making.

1.2.3 Definitions and a Lower Bound

In this section, we give a definition of compaction and show a preliminary hardness result that compaction is NP-hard. The proof is simple and easy to understand. We use the result here just to illustrate the difficulty of the problem. We will return to the same topic in Chapter 9 where we show that the problem is PSPACE-hard.

We define the *compaction* problem as a two-dimensional coordinated motion planning problem for a set of objects such that the area of the bounding convex hull or the bounding rectangle of the objects is minimized (see Figure 1.4). We consider only the case where the objects are rigid bodies and are represented as polygons. A *translational compaction* problem is a compaction problem in which the polygons are only allowed to translate. A *rotational compaction* problem is a compaction problem in which the polygons are allowed to translate and rotate. In contrast with the strip compaction problem defined in the next paragraph, we will refer the compaction problem defined in this paragraph as the *general compaction* problem.

Since our interest in studying compaction arises from strip packing applications (specifically, marker making) in which the motion of the objects is restricted to a strip of fixed width and arbitrary length, we define *strip compaction* to be a special form of the compaction in which the motion of all the polygons is confined to a bounding rectangle whose top, left and bottom sides are fixed and whose right side can move freely. The goal is to minimize the length of the bounding rectangle (see Figure 1.5).

As far as the complexity of motion planning is concerned, it is easy to show that general compaction is at least as hard as the strip compaction. We prove this fact by the reduction

[1] In Chapter 4, we will show how to implement this strategy.

[2] Some markers we have successfully compacted contain more than 400 pieces

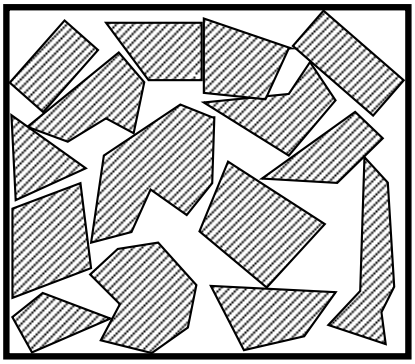


Figure 1.4: General compaction: minimizing the area of the bounding rectangle.

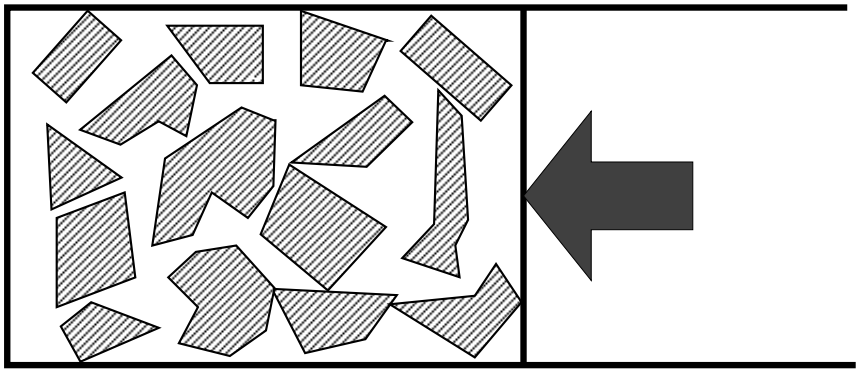
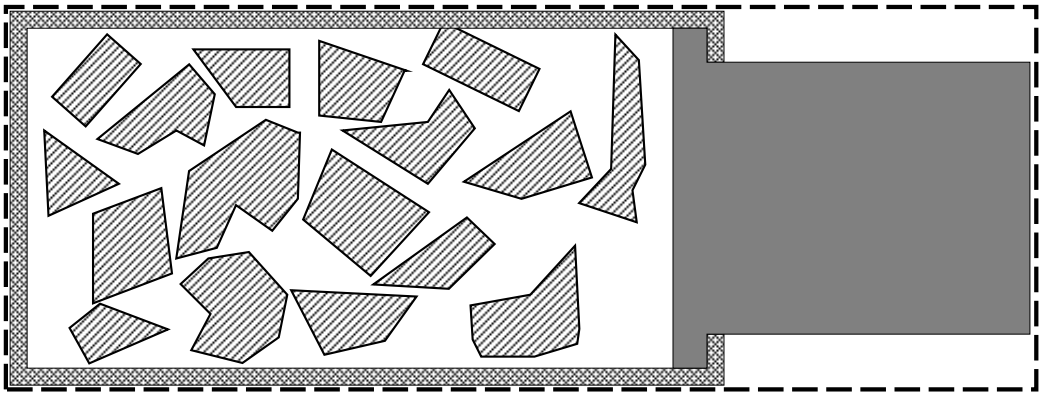


Figure 1.5: Strip compaction: minimizing the length of a fixed width bounding rectangle.



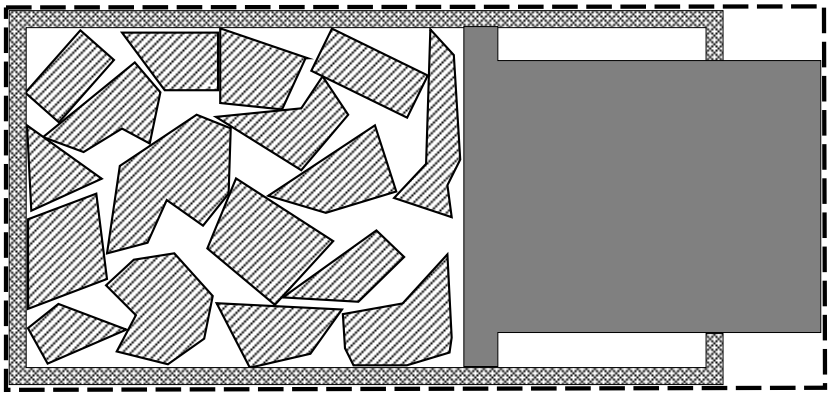


Figure 1.6: Reduction of strip compaction to general compaction.

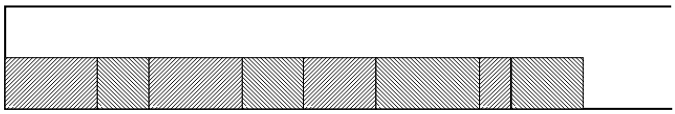


Figure 1.7: Reduction of PARTITION to strip compaction.

of strip compaction to general compaction shown in Figure 1.6. Given a set of polygons as an input to strip compaction, we make it an input to general compaction by adding a rectilinear "container" polygon and a "piston" polygon that jointly enclose the original set of polygons. The piston polygon can slide horizontally but has no vertical freedom. It is clear that the strip compaction problem has a minimum length if and only if the general compaction problem has a minimum area bounding box or bounding convex hull.

Therefore, a lower bound for strip compaction naturally establishes a lower bound for general compaction. Next, we show strip compaction is NP-hard.

Theorem 1.1 *The strip compaction problem is* NP-hard *even when all the polygons are rectangles and can only translate.*

Proof: We reduce the NP-hard PARTITION problem to strip compaction. The PARTITION problem is defined as follows: Given a set S of integers, decide if S can be partitioned into two subsets S_1 and S_2 such that the sum of the elements of S_1 equals the sum of the elements of S_2. We reduce PARTITION to compaction by the construction shown in Figure 1.7. For an instance $\{a_1, a_2, \ldots, a_n\}$ of PARTITION, we build rectangles of height 1 and length a_i, $(i = 1, \ldots, n)$. We put the rectangles into a container of height 2. PARTITION has a solution if and only if the blocks can be compacted to the length $\sum a_i/2$. □

In the rest of the thesis, we will refer the strip compaction problem as *the compaction problem* or simply *compaction*.

1.2.4 Related Work

As far as we know, no efficient algorithm has been previously published on how to compact non-convex polygons (see [SP92a] for a survey of current results). In industry, several CAD firms have tried some "one piece at a time" methods, which are not successful in practice.

VLSI Design

Previous research efforts in compaction are largely concentrated in the field of VLSI design [AKS90] [Len84] [Map90] [MFS87] [SSVS86] [Won85] [Mal90]. The idea of compacting a VLSI circuit layout is quite natural since it allows more transistors to be packed in the same area.

The layout problems in VLSI design are generally quite different from polygon nesting problems. The basic objects in a VLSI layout are rectangles, which represent gates, and variable-length rectilinear[3] wires that link the gates. There are minimum separation conditions imposed on the location of the rectangles. For example, one such condition might be that the lower boundary of a rectangle be three distance units above the upper boundary of another rectangle.

Finding the minimum area compaction for a VLSI layout is NP-complete. The general compaction problem is usually decomposed into two one-dimensional problems in the x and y direction in which rectangles can only slide horizontally or vertically. By exploiting the special forms of the inequalities which express the constraints (each inequality has two variables), one-dimensional compaction can be reduced to calculating the longest (or shortest) path in a constraint graph [Len84] [Won85]. In [AKS90], a sweepline type of algorithm to solve the one-dimensional compaction of layout consists of only rectangles (*i.e.* no wires). Other techniques include simulated annealing [MFS87] and "zone-refining" [SSVS86] – a technique which simulates the crystal refinement process.

The above mentioned techniques depend on the simple shape (rectangular or rectilinear) of the objects in order to build constraints on the relative positions of the objects. Most of them cannot make further improvement to a layout in which the left and right boundaries of the rectangles are touching or have met the minimum separation condition. We found that the techniques are not readily extendible to non-rectangles and not capable of squeezing out the last few fractions of a percent of material waste in an already tightly packed layout.

Physically Based Simulation

Physically based simulation methods provide a plausible approach to compaction. These methods simulate the pieces as rigid bodies and apply a set of forces to move the bodies in the desired direction. A local minimum is reached when the bodies cannot be moved further.

[3] *Rectilinear* means that the edges are parallel to the coordinate axes. Sometimes the term *orthogonal* is used instead.

These simulation methods fall into two types: *spring* model and *contact force* model. The difficulty we have found with physically based simulation methods is that these algorithms run very slowly.

Spring model methods (also called *penalty methods*) [MW88] [PB88] allow the pieces to inter-penetrate slightly. A restoring force is determined by the amount of inter-penetration. From these forces, the motion of the pieces is computed by numerical integration. This integration is carried out by cutting time into steps. Small steps are required to ensure accuracy and numerical stability of the integration; however, small steps means many steps and greater computational cost. Furthermore, strongly enforcing the non-overlapping condition implies a strong restoring force, and stronger forces require smaller steps. It is generally very difficult to choose an acceptable step size, especially when many different forces are involved. Layout compaction involves hundreds of frequently changing contacts, each change of contact requires a new numerical integration, and each numerical integration requires many steps. Hence, under the spring model, solving a compaction problem requires many steps and a long running time. Due to its long running time, the spring model is not suitable for compaction.

Contact force model methods (also called *analytical methods*) [Bar89] [BB88] have been recently studied in computer graphics. Contact forces are introduced at contact points to prevent inter-penetration. A system of differential equations is solved to find a set of consistent contact forces. The bodies then move with constant velocities in accordance with the contact forces. A collision detection algorithm is used to find the time interval until the next collision occurs. The simulation proceeds from time interval to time interval until the local minimum is reached. Contact force methods have the advantage that the time intervals are much larger and more easily determined than the time steps of the spring model methods. However, there still are difficulties. Solving the system of differential equations for determining the contact forces is time-consuming for problems of non-trivial size.

Even setting these difficulties aside, our experience shows that in the case of many contacts the local minimum is reached only after many time intervals because the pieces tend to "rattle" against each other.

Another problem of contact force method is that it is NP-hard to resolve vertex-vertex contacts. Basically, the problem is that when two vertices from two polygons are in contact, one has to choose which way the vertices "pass" each other. Does one go to the other's left or right? There are situations where the polygons are "locked" with each other. We

can show that it is NP-hard to determine if there is a simultaneous choice of motion for each pair of vertices that "unlock" the layout. This problem persists for the velocity and position based compaction algorithm we developed in this thesis. However, the occurrence of a locked layout is extremely rare and we believe it does not have much impact in practice.

Robot Motion Planning

In the area of robot motion planning, research has been concentrated on how to plan the motion of a single object in an environment consisting of polygonal or polyhedral obstacles, a problem referred to as the *generalized movers' problem* [Rei79] [Rei87] [SS90]. The problem is shown to be PSPACE-hard by John Reif [Rei79] [Rei87] and the current best algorithm for it is that of Canny which runs in single exponential time [Can87].

The research on planning simultaneous motion of many objects has been limited to a few negative results. The best known of them is the PSPACE-hardness result obtained by Hopcroft, Schwartz and Sharir on the Warehouseman's problem. The result states that planning the coordinated motion for a set of rectangles inside a rectangular container is PSPACE-hard.

However, the techniques developed in the context of single object motion planning, especially the configuration space approach [LPW79], prove to be extremely useful in compaction. For planning the motion of a polyhedron in three dimensions, the configuration space is the six-dimensional space of all the possible values for the three positional parameters and the three rotational parameters. The configuration space is partitioned into connected components classified as *free space* or *forbidden regions*. Each point in a free space gives a configuration (position and orientation) of the polyhedron which is clear of obstacles. The boundaries between free spaces and forbidden regions are generally high degree algebraic surface patches. However, if the polyhedron is only allowed to translate and the environment consisting of only polyhedral obstacles, then the boundaries of free spaces are polyhedral surfaces.

1.3 Contribution

1.3.1 A Position-Based Optimization Model

The main contribution of this thesis is the first fast and practical algorithm for compacting a layout consisting of non-convex polygons. The algorithm has been implemented

and tested on production quality human generated markers. Our experiments show that even though production quality markers are very tightly packed, almost all of them can be improved in efficiency by our compaction algorithm. The algorithm runs in almost real time in practice. Thus, it solves industrially significant problems and fills a gap in the research of compaction algorithms.

The key to the algorithm is a position-based optimization model which is essentially a linear programming formulation of coordinated motion planning of many translating objects under a set of "forces". These forces are defined by a potential function on the positions of the objects. The model is based on the configuration space approach [LPW79] in robot motion planning and a new *locality heuristic*. The locality heuristic enables us to find a maximal convex subset of the free space and makes the direct calculation of positions of the polygons possible via linear programming.

The position-based optimization model is highly flexible. Various types of layout tasks can be accomplished by specifying appropriate forces applied on the polygons and by adding and/or modifying constraints. For example, to solve the compaction problem itself, we can add an imaginary "piston" piece on the right of the layout and specify a leftward force applied to the piston piece. In the trim placement phase, we often need to enlarge a gap in order to put more pieces or larger pieces in the gap. Such a task can be accomplished by specifying forces that push away the pieces surrounding a gap. Other tasks, such as simulating objects in a gravitational field, fixing some polygons while allowing others to translate or even eliminating the overlap among the pieces in a marker, can be accomplished directly by applying the optimization model.

It is interesting to notice that the position-based model is capable of eliminating overlaps in a marker. This is because the slack variables for each non-overlapping constraint in our model exactly correspond to the amount of overlap between pairs of polygons. We realized that we could extend the model by making these variables explicit in the constraints and adding them to the original objective of our model. The extended position-based model can eliminate overlaps, apply forces, and even float pieces away from each other (negative overlap) in any desired combination.

The overlap-elimination algorithm has been applied in database driven automated marker generation. The underlying idea is to collect high quality markers generated by experienced human marker makers into a database. When a new marker making order comes in, we search the database for a similar marker. Then we match the pieces and do a piecewise

substitution of the corresponding pieces. The substitution process will inevitably introduce overlaps. By applying separation and leftward compaction, we can generate a marker of sufficient quality automatically. A CAD company in the textile industry is now vigorously pursuing the database driven automated marker making idea.

We have studied several other extensions to the position-based optimization model. We offer two methods, *relaxation* and *linearization*, to extend the model to handle small rotations (tilts). These methods are the first known attempts for solving the rotational compaction problem. In another extension, we replace the linear programing formulation used in the model with a mixed integer programming (MIP) formulation. We use the MIP formulation to find the optimal solution of the two-dimensional strip packing problem and to solve the multiple containment problem, the problem of placing non-convex polygons in a non-convex container.

1.3.2 A Velocity-Based Optimization Model

Another contribution of this thesis is a velocity-based model for performing compaction tasks. This model is more in line with the physically based simulation methods using contact forces. However, by removing the concepts of mass and force and directly computing velocities, this model yields a new, more efficient, time-interval-based simulation technique for translational compaction. We had expected that direct computation of velocities would allow our algorithm to solve the compaction problem in a reasonable amount of time. Unfortunately, when compacting tightly packed layouts, our simulation method runs two orders of magnitude slower than the the compaction algorithm derived from the position-based model. Even though our new, velocity-based simulation techniques reduced the running time by at least an order of magnitude over previous physical simulation methods, the algorithm still took hours to run on typical markers on a 28MIPS workstation because of the number of times it had to recompute velocities. This can be viewed as a important negative result that demonstrates the inherent limitation of the physically based approach.

Nevertheless, our velocity-based model provides an interesting alternative to the current physically based simulation methods. For example, formulating the non-penetration constraints is quite involved in physically based simulation methods. In our method, however, the non-penetration constraints are purely geometric and are very easy to set up. In addition, we have observed from experiments that our simulation method is quite efficient in bringing loosely packed layouts to a relatively tight status. We believe our method will give

better performance in the simulation applications, such as animation, for which obeying physical laws is not essential.

Both the position-based and the velocity-based models apply to general polygonal shapes. Therefore, the algorithms can be applied not only to markers of all types of garments but also to polygon nesting problems in other industries.

1.3.3 Complexity

In this thesis, we also study the lower bound of the compaction problem. We show compaction to be PSPACE-hard by a reduction from the PSPACE-hard symbol transposition problem through the well-known Warehouseman problem [HSS84]. In addition, we give the first explicit constructions that show an exponential number of moves is necessary in the worst case for the compaction problem. More specifically, we give two such constructions. The first construction shows that an exponential number of moves is required even if the objects involved in compaction are as simple as rectangles and U-shaped objects. The second construction is more contrived. But it shows that finding a local minimum of compaction using the compaction algorithm in Chapter 4 requires an exponential number of moves.

1.4 Organization of the Thesis

The rest of the thesis is organized as follows.

Chapter 2 presents a new, velocity-based compaction algorithm. This algorithm simplifies the existing contact-force based physical simulation approach. The algorithm solves a velocity-based optimization model to generate a set of velocities for the set of polygons in a layout and then simulates motion until a new contact occurs. We discuss its long running time for tightly packed markers and why that is an inherent drawback of physically based methods.

Chapter 3 gives background knowledge about configuration spaces Minkowski sums and Minkowski differences, which play a central role in the position-based optimization model. When planning the translational motion of a polygon in a environment consisting of a single polygonal obstacle, the configuration space can be obtained by calculating the Minkowski sum of the polygon and the obstacle. We show that for certain classes of polygons (star-shaped), the Minkowski sum can calculated very efficiently. The position-based model can be applied to non-star-shaped polygons by decomposing them into star-shaped polygons.

Chapter 4 develops the position-based optimization model which directly determines new positions for the polygons. We then give several algorithms based on the model to solve the compaction problem and perform some layout related tasks. As a result of the direct calculation of positions, the algorithms based on the model do not use simulation nor do the have time as an explicit parameter.

Chapter 5 defines the problem of separating overlapping polygons, studies its complexity, and shows how the position-based model yields a separation algorithm that finds a locally optimal solution to this problem. We next demonstrate that by combining the separation algorithm and a database of human generated markers, we can have an automated marker making system that very quickly generates markers close to human performance.

Chapters 6, 7 and 8 contain extensions to the position-based optimization model. Chapter 6 develops two rotational compaction algorithms. Chapter 7 introduces variables which represent distances between polygons into the position-based model. We use these distance variables to control the amount of overlaps between polygons or to float polygons away from each other. Chapter 8 introduces a mixed integer programming (MIP) formulation which replaces the linear programming (LP) formulation in the original position-based model. Then we show that the two-dimensional strip packing and the multiple containment problems can be reduced to a mixed integer program through the MIP formulation.

Chapter 9 studies the complexity of the compaction problem. We show that compaction is PSPACE-hard even for rectangles. We then give two explicit constructions to show that coordinated motion planning in general and compaction in particular require an exponential number of moves in the worst case.

Finally, Chapter 10 concludes the thesis by summarizing the results and discussing open problems and directions for future work.

Chapter 2

Compaction Using a Velocity-Based Optimization Model

In this chapter, we give an optimization model for determining the velocities of rigid bodies under a set of forces, and we use this model to create a velocity-based compaction algorithm. Like the previously proposed physically based simulation methods (Section 1.2.4), our algorithm uses time intervals to advance the simulation steps. However, by ignoring the dynamic properties of the rigid bodies, our simulation method is simpler and faster than the previous simulation algorithms using the contact force model.

2.1 High Level Description of the Compaction Algorithm

Let us for the moment view the polygonal pieces in a marker as frictionless rigid bodies and simulate their motions within a rigid rectangular container. We see that compaction can be performed by applying forces to the pieces in the desired directions. For example, to shorten the length of a layout, we can add a "piston" piece on the current right boundary of the layout and by assign a leftward force to the piston piece. Hence, it is quite natural to apply physically based simulation methods for compaction.

As reviewed in Section 1.2.4, there are two basic approaches to physically based simulation methods: the spring model and the contact force model. Spring model based methods have numerical problems caused by the stiff differential equations associated with the spring

forces introduced between contact points. These methods are not seemed to be suitable to our application. We will therefore concentrate on the contact force model.

In contact force based algorithms, the simulation process proceeds according to time intervals. The control loop of a typical contact force based simulation algorithm consists of three steps: (1) collision detection, (2) building non-penetration constraints and (3) solving for a set of new motions. Let t_{k-1} be the starting time of the current simulation step. A contact force based simulation first calculates the next collision time t_k using a collision detection algorithm according to the current motions of the objects.[1] It then advances the simulation to t_k using the current motions. Next, a set of non-penetration constraints are built for the objects that are in contact at t_k. The constraints ensure that the new motions calculated will not cause inter-penetration among the objects currently in contact. The algorithm then concludes the current simulation step by updating the motions of the objects with a set of new motions calculated from the non-penetration constraints. The end time t_k for the current simulation step becomes the starting time of the next step. The term *time interval* refers to the time between two simulation steps, which is the time between two consecutive collisions.

Our velocity-based algorithm uses the same general control loop as the contact force based simulation algorithms for rigid bodies. However, we replace the contact force model with a *velocity-based optimization model* which allows us to calculate velocities using linear programming. Our model is based on the observation that, for compacting a two-dimensional layout, the most important property of the objects is rigidity. Rigidity ensures that no inter-penetration can occur among the pieces. If translation is the only motion allowed, then the rigidity condition for an object is that every point on the object has the same velocity. As shall be shown later, the non-penetration condition for two touching object can be expressed as a set of linear constraints on the velocities of the two objects. These constraints plus an objective that maximize the velocities in the direction of the forces comprise a velocity-based optimization model which replaces the Newtonian physics based differential equation formulation to solve for a new set of velocities.

We now give the high level description of the velocity-based compaction algorithm, and then supply several low level details in the following sections.

Step 1. (Initialization) The input is a layout consisting of N polygons and a constant "force" f_i, $1 \leq i \leq N$, for each polygon. The forces represent the desired directions

[1] The initial motion at t_0 might be simply a leftward motion of the "piston" piece.

we want the polygons to move. Assign a "velocity" variable $\mathbf{v}_i = (v_{ix}, v_{iy})$ to polygon i, $1 \leq i \leq N$. The magnitudes of the velocities of the movable polygons are bounded by a constant c, that is, $-c \leq v_{ix}, v_{iy} \leq c$ for $1 \leq i \leq N$. Initially, each v_i is set equal to f_i and scaled down to satisfy the magnitude constraint.

Step 2. Let the current time be t_{k-1}. Find the next collision time t_k using a collision detection algorithm to be described later. Let $\bar{t} = t_k - t_{k-1}$ be the time interval between the current time and the next collision time. If Steps 3 and 4 have been executed at least once and if $\bar{t}$ is less than a predefined threshold, terminate the algorithm. Otherwise, translate polygon i $(1 \leq i \leq n)$ by $\bar{t}\mathbf{v}_i$ and go to Step 3.

Step 3. Identify all the vertex-edge touching pairs at time t_k. A vertex and an edge form a *vertex-edge touching pair* if the vertex and the edge are from two different polygons and the vertex is incident on the edge.

For each of the vertex-edge touching pairs (A, BC), where A is a vertex of polygon P_i and BC is an edge of polygon P_j $(i \neq j)$, generate a constraint on the velocities of P_i and P_j such that, if P_i and P_j move with velocities satisfying the constraint, then the two edges of P_j adjacent to A cannot overlap BC. As will be shown later, the constraint is a linear inequality in $\mathbf{v}_i$ and $\mathbf{v}_j$.

Step 4. Set up a linear program that maximizes $\Sigma \mathbf{f}_i \cdot \mathbf{v}_i$. The linear program is subject to the non-penetration constraints built in Step 3 and the upper and lower bounds on the magnitude of the velocities as specified in Step 1. This objective and constraint set form our *velocity-based optimization model.* Solving the linear program yields a new set of velocities. Goto Step 2.

2.2 Collision Detection

2.2.1 Collision Detection For Two Translating Polygons

Given a pair of translating polygons P and Q, we first note that if an edge e_P of P and an edge e_Q of Q are going to collide, the first collision between e_P and e_Q cannot occur solely on two interior points of P and Q. In other words, at the earliest collision time, at least one vertex is involved. Therefore, we can capture the earliest collision time by only considering the vertices of P and the edges of Q and vice versa.

Let the velocities of P and Q be $\mathbf{u}$ and $\mathbf{v}$ respectively. Let A be a vertex of P and BC an edge of Q. Let the starting time of the current simulation step be t_{k-1}. We define function *CollisionTime*(A, BC, $\mathbf{u}$, $\mathbf{v}$, t_{k-1}, t_k) to return the collision time between A and BC if the collision occurs before t_k. Otherwise, *CollisionTime*(A, BC, $\mathbf{u}$, $\mathbf{v}$, t_{k-1}, t_k) just returns t_k.

Note that the collision time between A moving with velocity $\mathbf{u}$ and BC moving with velocity $\mathbf{v}$ is the same as the collision time between a static BC and A moving with relative velocity $\mathbf{u}-\mathbf{v}$. Thus, there is a collision between A and BC within time interval $\bar{t} = t_k - t_{k-1}$ if and only if the line segment AA' intersects the line segment BC, where $A' = A + \bar{t}(\mathbf{u}-\mathbf{v})$ is the new location of A relative to BC at time t_k. Therefore, collision detection for a moving vertex and a moving edge within a pre-specified time is reduced to the problem of testing whether two line segments intersect and finding the intersection point. Milenkovic [Mil88] proposed a numerically robust algorithm for line segment intersection which is used in the implementation of our algorithm.

2.2.2 A Collision Detection Algorithm

Let t_{k-1} be the starting time of the current simulation step. The algorithm for finding the next collision time among the objects is as follows.

Algorithm for Collision Detection.
Input: a layout L consists of N polygons $P_1, P_2, \ldots, P_N$.
The velocities of these polygons are $\mathbf{v}_1$, $\mathbf{v}_2$, ..., $\mathbf{v}_N$.
for every pair of polygons $P = P_i$ and $Q = P_j$ $(1 \le i \le N, 1 \le j \le N, i \ne j)$
for every vertex A in P and every edge BC in Q
$t_k \Longleftarrow$ *CollisionTime*(A, BC, $\mathbf{v}_i$,$\mathbf{v}_j$, t_{k-1}, t_k);

During the execution of the algorithm, the variable t records the earliest collision time calculated so far. Initially, t_k is assigned T where T is a predefined value. For simplicity, we give T a relatively large value within which a collision is likely to occur. The next collision time t_k is then reduced successively when *CollisionTime*(A, BC, $\mathbf{v}_i$,$\mathbf{v}_j$, t_{k-1}, t_k) finds an earlier collision time. When the algorithm terminates, t_k holds the time of the earliest collision.

Assuming the maximum number of vertices in a polygon is M, the collision detection algorithm clearly runs in $O(N^2M^2)$ time. The algorithm is simple and easy to implement.

Note that if there is no collision occurs within time interval T, then it is safe to advance the simulation by T. That is, assuming the velocity of polygon P_i is $\mathbf{v}_i$ for $1 \leq i \leq N$, translated every polygon P_i to its new position $P_i + T\mathbf{v}_i$ $(1 \leq i \leq N)$ and resume the simulation at that point.

The algorithm, although simple, includes the steps necessary for a collision detection algorithm. The running time can be improved in various way. For example, for polygons P_i and P_j, we can first test whether their bounding boxes collide within time interval T. If not, then we can omit the collision detection for P_i and P_j.

Another way of improving the running time is that instead of checking all $O(N^2)$ pairs of polygons, we can first eliminate the pairs that cannot collide in time T and only check the remaining pairs. We eliminate a pair by testing their *extended bounding boxes*. For a polygon P_i, its extended bounding box B_i is the minimal area axis parallel rectangle that includes both P_i and $P_i + \mathbf{v}_i T$. Obviously, if B_i and B_j $(i \neq j)$ do not intersect, P_i and P_j cannot collide within T. By using a sweepline algorithm [PS85], we can identify all K pairs of overlapping B_i's in $O(N \log N + K)$ time. If we choose T to be small, which we do for a tightly packed layout, then only the extended bounding boxes of a pair of "neighboring" polygons can intersect. In that case, K is linear in N since the "neighboring" graph of a layout is a planar graph. Hence, the running time of the sweepline algorithm is actually $O(N \log N)$. And since for each "neighboring" pairs, we check $O(M^2)$ vertex-edge pairs at most, the overall running time of the collision detection algorithm can be reduced to $O(N(\log N + M^2))$ in practice.

We can further improve the running time for determining when the first contact between P_i and P_j occurs from $O(M^2)$ to $O(M \log M)$, again using a sweepline algorithm. The sweepline is parallel to $\mathbf{v}_i - \mathbf{v}_j$. As we sweep through each vertex A of P, we determine the smallest t such that $A + t(\mathbf{v}_i - \mathbf{v}_j)$ is on an edge of Q. Similarly, as we sweep through each vertex B of Q, we determine the smallest value of t such that $B + t(\mathbf{v}_j - \mathbf{v}_i)$ touches P. By maintaining a dynamic list of active edges of P and Q along the sweepline, we can determine each value of t in $O(\log M)$ time. This improvement reduces the overall running time of the collision detection algorithm to $O(N(\log N + M \log M))$.

2.3 Finding Vertex-edge Touching Pairs

2.3.1 Type of Contact

In our application, the input to the compaction algorithm is a non-overlapping layout. That is, in the initial configuration no point of a polygon P_i can be in the interior of another polygon P_j, for $i \neq j$, $1 \leq i, j \leq n$. It follows that the next collision can only occur on the boundaries of the polygons.

For two polygons that are in contact at the next collision time t_k, the contact among the points on the boundaries of the two polygons can be classified into the following types:

- vertex-edge contact.
- edge-edge contact.
- vertex-vertex contact.

As shown in Figure 2.1 and Figure 2.2, an edge-edge contact can be broken into two vertex-edge contacts; and a vertex-vertex contact can be broken into four vertex-edge contacts. So we only need to concentrate on finding all the vertex-edge contacts, *i.e.* the vertex-edge touching pairs.

2.3.2 A Simple Algorithm

A simple algorithm for finding all the vertex-edge touching pairs is to check every pair of polygons and, for each pair of polygons, test the vertex-edge incidences for every vertex of one polygon against the edges of the other polygon. This algorithm is essentially the same as the simple collision detection algorithm and runs in $O(N^2M^2)$ time.

2.3.3 A Sweepline Algorithm

Given a layout of N non-overlapping polygons we can find all the vertex-edge touching pairs in the layout by calculating the arrangement of the NM edges in the layout explicitly [Ede87]. The pairs of intersecting edges in this arrangement has the property that their intersection occurs at the place where one vertex of an edge is incident on the other edge. Hence, the pairs of intersecting edges in the arrangement forms the vertex-edge touching pairs.

By Chazelle and Edelsbrunner's result [CE88], the arrangement of the $O(NM)$ edges can be computed optimally in $O(NM \log(NM) + K)$ time where K is the total number

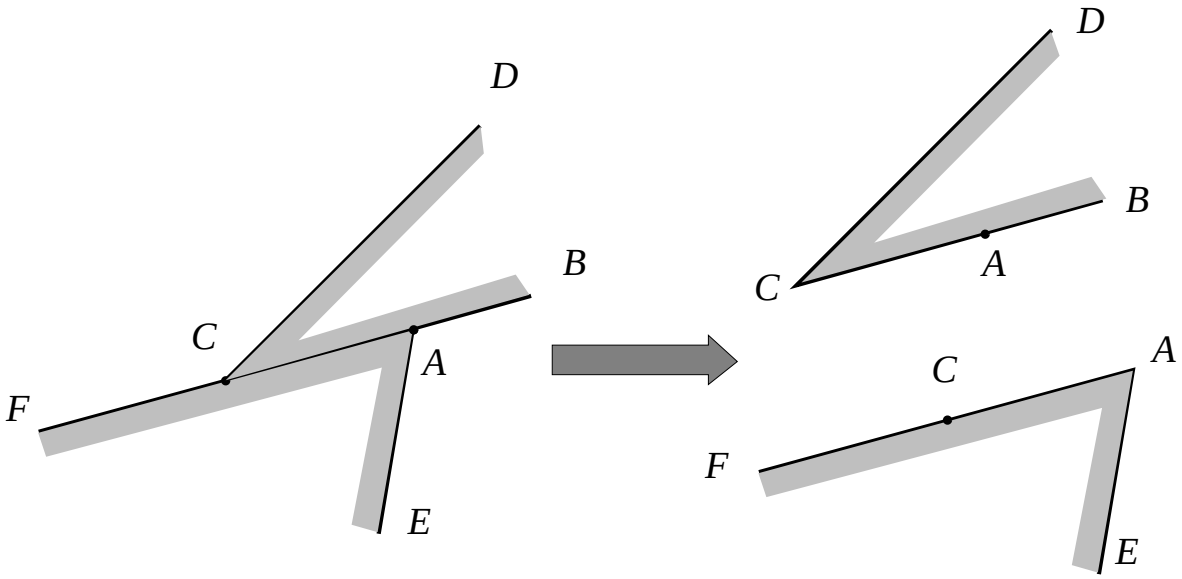


Figure 2.1: The non-penetration constraint for an edge-edge contact can be broken into two vertex-edge contacts: (A, CB) and (C, AF)

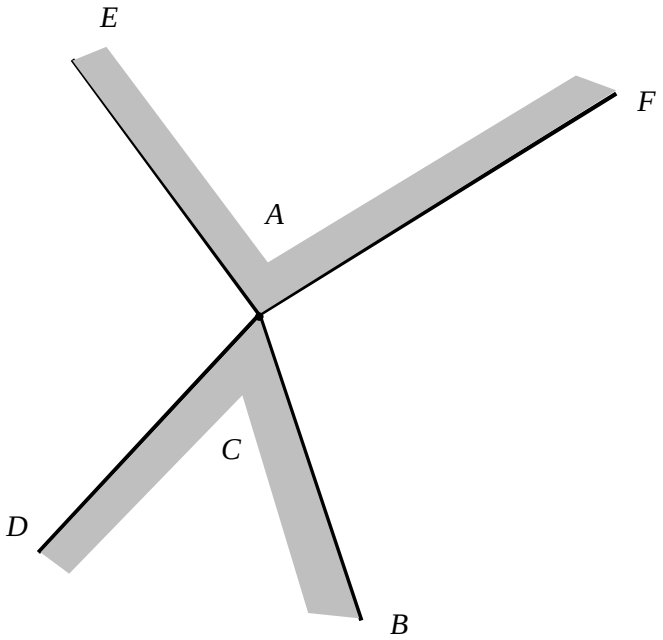


Figure 2.2: The non-penetration constraint for an vertex-vertex contact between $\angle FAE$ and $\angle BCD$ can be broken into four vertex-edge contacts: (A, BC), (A, CD), (C, FA) and (C, AE).

intersecting edge pairs which, in our case, are vertex-edge touching pairs . Assume that there is no more than a constant integer $c > 0$ number of vertices incident on one point, there is at most $O(NM)$ vertex-edge touching pairs. Hence the algorithm runs in $O(NM \log(NM))$ time.

2.3.4 An Alternative Algorithm

The previous two algorithms run separately with the collision detection algorithm. The two algorithms assume that the collision detection algorithm has run to the end and the simulation has been advanced by a time interval $\bar{t}$ found by the collision detection algorithm. An alternative method for finding vertex-edge touching pairs is to augment the collision detection algorithm of Section 2.2. This method generates a list of vertex-edge touching pairs during the execution of the collision detection algorithm.

Assume t_k is the next collision time calculated when the collision detection algorithm terminates. If vertex A of polygon P and edge BC of polygon Q yield the earliest collision time so far, then (A, BC) is a candidate vertex-edge touching pair at time t_k. We use a list L to record the all the candidate vertex-edge touching pairs. If t, the collision time between A and BC, is the earliest collision time recorded so far, then we include (A, BC) in L. Let t' be the collision time between D and EF, the next vertex-edge pair to be checked. If $t' < t$, then (A, BC) and the vertex-edge pairs currently in L cannot be vertex-edge touching pairs at t_k. Therefore, we delete the elements in L and insert the pair (D, EF) into L. If $t' = t$, then (D, EF) is appended to L as a candidate. In the last case, which is $t' > t$, L is not changed. When the collision algorithm algorithm terminates, L contains all the vertex-edge touching pairs at time t_k. Since this algorithm is part of a collision detection algorithm, it runs in the same time as the collision detection algorithm, namely $O(N(\log N + M \log M))$.

2.4 Non-penetration Constraints

The purpose of the non-penetration constraints is to ensure that, starting from the next collision time, a vertex-edge touching pair cannot inter-penetrate each other when the simulation advances.

Let A be a vertex of polygon P that is touching edge BC of polygon Q at the collision time. The points on the boundary of a polygon are ordered counterclockwise, i.e. the interior of polygon Q is on the left side of the directed edge BC. Polygon P is to be moved with velocity vector $\mathbf{u}$ and polygon Q with velocity vector $\mathbf{v}$. After a time interval $\bar{t}$, A is

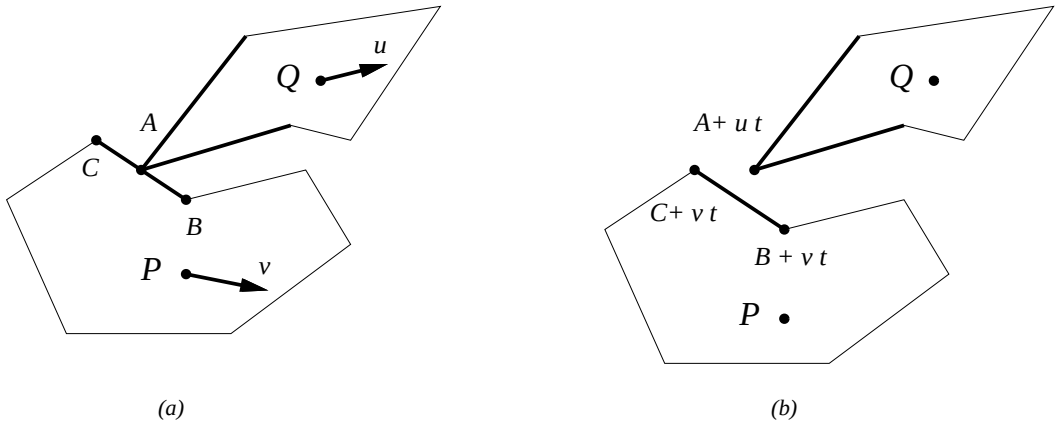


Figure 2.3: (a) The vertex A of P touches the edge BC of Q. P moves with velocity u and Q moves with velocity v. (b) The position of P and Q after time interval $\bar{t}$.

moved to $A' = A + \mathbf{u}t$ and edge BC is moved to $B'C'$ where $B' = (B + \mathbf{v}t)$ and $C' = (C + \mathbf{v}t)$ (See Figure 2.3). The condition that A' does not penetrate $B'C'$ requires that A' stay on the right-hand side of the directed edge $B'C'$, i.e. A', C' and B' should be in counterclockwise order. By Lemma A.3 (see Appendix A), this condition can be expressed as:

$$B'A' \times B'C' \geq 0$$

that is

$$((A + \mathbf{u}t) - (B + \mathbf{v}t)) \times ((C + \mathbf{v}t) - (B + \mathbf{v}t)) \geq 0$$

This simplifies to

$$(A - B) \times (C - B) + (A - B) \times (\mathbf{u} - \mathbf{v})t \geq 0$$

and

$$BA \times BC + BA \times (\mathbf{u} - \mathbf{v})t \geq 0$$

since A was touching BC,

$$BA \times BC = 0$$

and since $t > 0$, the non-penetration constraint becomes:

$$AB \times (\mathbf{u} - \mathbf{v}) \geq 0 \tag{2.1}$$

which expands to:

$$(A_y - B_y)u_x - (A_y - B_y)v_x - (A_x - B_x)u_y + (A_x - B_x)v_y \geq 0$$

Thus, we have shown that the non-penetration constraint is a linear inequality on the velocities $\mathbf{u}$ and $\mathbf{v}$.

In the above formulation, vertex A must not be incident on vertex B, since in that case the constraint we have just derived will be a null constraint. When A is incident on B, we can derive the non-penetration constraint from:

$$B'C' \times A'C' \geq 0$$

which expands and then simplifies to

$$(\mathbf{u} - \mathbf{v}) \times AC \geq 0 \tag{2.2}$$

2.5 Bounds on the Velocities

In the algorithm, we have specified upper bounds on the magnitude of the velocities. These bounds are necessary because from the previous section we see that the non-penetration constraints derived forms homogeneous system of inequalities. That is, in the constant terms of the constraints are all zeros. It is possible that optimizing the the objective function $\Sigma f_i v_i$ can cause the velocities to increase without bound. On the other hand, simultaneously scaling of all the velocities will not affect the behavior of the simulation. Specifically, if the velocities are all scaled up by a factor of F, then the time interval $\bar{t}$ between the current time and the next collision will be scaled down by a factor of F accordingly. Therefore the net effect is still that each of the polygons is translated by the same amount.

2.6 Non-Penetration Constraints for Vertex-Vertex Contacts

We have seen that the a vertex-vertex contact is broken into four vertex-edge contacts. However, as we will see, it is not necessary to build non-penetration constraint for each of vertex-edge contacts. A constraint is necessary only if the vertex and the edge form a *vertex-edge supporting pair*. Let A be a vertex of polygon P and and BC be an edge of polygon Q. Let the two edges incident to A be FA and AE. We say that A and BC form an vertex-edge supporting pair if among the points in edge FA and AE, A has the smallest directed distance to L_{BC}, the line that contains BC (see Figure 2.4). In other words, if

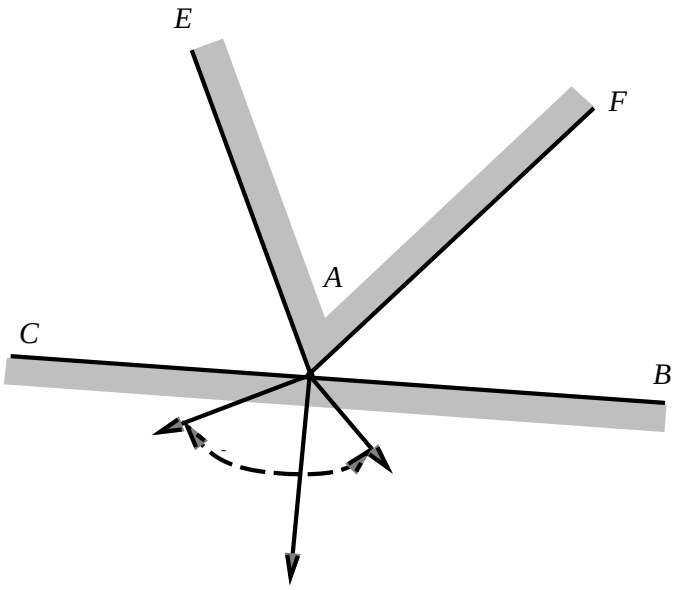


Figure 2.4: A vertex-edge supporting pair.

$\angle EAF$ is moved sufficiently far way and moved back towards L_{BC}, then A is the first point to hit L_{AB}. We say A supports BC if A and BC form a supporting pair. If we draw the outward normals of FA and AE and the inward normal of BC starting from A, then the inward normal of BC is included in the "cone" of angle $\leq$ 180 degrees bounded by the outward normals of FA and AE.

If A and BC do not form a vertex-edge supporting pair, then force A stay outside BC does not guarantee non-overlapping condition of FA and AE with BC. Therefore, it is not necessary to build a non-penetration constraint for A and BC.

From the definition, it immediately follows that a concave vertex cannot support any edge. In a non-overlapping layout, there cannot be concave vertex contact another concave vertex.

For a convex-concave vertex contact pair, the convex vertex and the two edges of the concave vertex form two vertex-edge supporting pairs and both of them need to be included when building non-penetration constraints.

For convex-convex vertex contact pair, there can be only two vertex-edge supporting pairs among the four vertex-edge contact pairs, and we only need to chose one of these two vertex-edge supporting pair to build a constraint. This fact can be seen more clearly from the configuration space and Minkowski sum point of view (see Chapter 3). We omit the explanation here.

As we discussed in Section 1.2.4, the choice between two of the vertex-edge supporting pairs for a convex-convex vertex contact pairs is arbitrary. Either one can be chosen to generate the non-overlapping constraint for P and Q. In making the choice, we want achieve two goals. The first goal to resolve the "locked" layout in which the polygons interlock with each. The second goal is to make the choice that can leads to a global optimum. However, we can show it is NP-hard to achieve either one of the two goals.

2.7 Performance

The simulation algorithm described thus far has been implemented on a 28mips Sun Sparc workstation and tested on real apparel markers. Comparing to the "strict" physically based methods, our simulation algorithm is simpler. The linear programming formulation which replaces the differential equations or quadratic programming formulations in the contact-force based physical simulation method should enable it to run much faster. The experimental results show that the algorithm is very efficient in converging an initially loosely packed layout to a relatively tight layout.

However, we discovered that our simulation algorithm has difficulty compacting tightly packed layouts. When polygons are tightly packed, the chances of collision increase dramatically. The time interval is usually very small, which causes numerical problems. Very frequently, the simulation goes through many small time interval steps before it can make a significant progress through a large time interval. Thus, the algorithm has not been successful in our application, which involves compacting tightly packed human generated markers. For a medium sized marker that contains around 40 pieces, the algorithm took more than 40 minutes.

The small time intervals have two causes. First, since there are many pieces, it is likely that there are two which are "just about" to come in contact. When they do, the simulation must stop and recompute. We could continue moving other pieces "forward in time" until they come in contact and possibly make more progress this way. However, the resulting simulation would have different "time zones" and thus would be even less similar to physical reality than our current algorithm. The second and more fundamental reason why the time interval is small is that even if there is only one piece, it is possible to have many small time intervals. Suppose piece P is in contact with the the "piston" piece on the right. As the piston moves leftward, P bounces back and forth between the left wall and the piston piece. Each bounce requires a time interval. In Chapter 4, we will give a *position-based*

model which will allow us to compute a final position for P and the piston in one step.

Although some steps of the simulation algorithm are carried out by straightforward algorithms whose running time can be improved, we determined that the long running time is mainly due to the small time intervals. The running time of the other steps of the simulation algorithm is insignificant compared to the large number of calls to the the linear program solver due to the small time intervals.

Chapter 3

The Theory of Minkowski Sum and Difference

The Minkowski sum and difference are powerful preprocessing tools for polygon intersection and containment problems. Using the Minkowski sum and difference respectively, we can convert a polygon-polygon intersection (overlap) query and a polygon-polygon containment query into point-in-polygon queries which allow us to achieve sub-linear query times. Although the Minkowski sum and difference has been used extensively in robot motion planning via the configuration space approach [Can87] [LPW79] [SS90], we have seen very few efforts in applying it to packing problems. Throughout our marker making project, we have demonstrated the greatly utility of the Minkowski sum and difference in packing problems. For example, the Minkowski sum plays a central role in the position based optimization model to be described in the next chapter. The Minkowski difference is essential in trim placement (see [DLM94] and Chapter 8).

In this chapter, we present the definition of the Minkowski sum and difference and discuss its properties and its applications in intersection and containment problems. We also examine algorithms for computing the Minkowski sums of different types of polygons. In particular, we prove a crucial property about the Minkowski sum of starshaped polygons which results in a significantly simplified algorithm for computing the Minkowski sum of starshaped polygons. We also show that similar properties hold for monotone polygons.

3.1 Background

3.1.1 Polygon Intersection and Containment Problems

We consider the following problems for two simple polygons P and Q.

Polygon intersection: Does P intersect (overlap) Q?

Polygon containment: Is P be totally contained in Q?

These two problems arise frequently in marker making (and in general, two-dimensional packing problems). Therefore, optimal or efficient solutions to the problems are of great interest.

Without preprocessing, there exist trivial linear-time lower bounds for both problems. These lower bounds are actually tight, according to the results of Van Wyk and Tarjan [TV88] and Chazelle [Cha90]. Van Wyk and Tarjan showed the polygon intersection problem is linear time reducible to polygon simplicity testing, that is, testing whether a polygon is self-intersecting. They further showed that testing the simplicity of a polygon is in turn linear time reducible to polygon triangulation, which is the problem of decomposing a polygon into non-overlapping triangles whose vertices are also vertices of the polygon. The final linear time bound for the polygon intersection problem is achieved by Chazelle's linear time triangulation algorithm [Cha90]. Therefore, we have

Theorem 3.1 *Polygon intersection can be solved in linear time.*

Using Theorem 3.1, we show next that polygon containment can also be solved in linear time.

Theorem 3.2 *Polygon containment can be solved in linear time.*

Proof: Let P and Q be two simple polygons. A necessary and sufficient condition for Q containing P is that at least one point of P is contained in Q and P and does not intersect the boundary of Q. Testing whether an arbitrary point of P is contained in Q takes linear time.

To test whether P intersects the boundary of Q, we identify a vertex u of P that has the largest y-coordinate. Extend a ray R vertically upwards from u. It is easy to determine in linear time the first point v of Q hit by R (if there is none, then P is not inside Q). Cut Q at v if it is not already a vertex. Add edges uv and vu to splice P and Q into one

loop: there are two copies of u and two copies of v. Move uv by ϵ to the left and vu by ϵ to the right, that is, separate the multiple copies of the vertices by 2ϵ. Only six edges are affected by changes in ϵ, and if ϵ is sufficiently small, then these edges do not intersect any others (unless P and Q have intersecting boundaries). This value of ϵ can be computed in linear time. Furthermore, for this value of ϵ, the spliced polygon will be simple (not self intersecting) if and only if P is interior to Q. The linear time bound follows from that testing the spliced polygon for simplicity takes linear time. □

Note that the linearity of both the intersection and containment problems depend on Chazelle's linear time triangulation algorithm. Chazelle's algorithm is a theoretical breakthrough in computational geometry. However, the algorithm is extremely complicated and it remains to be seen whether it can be made practical. It is also not clear whether the algorithm can be made numerically robust. In practice, Chazelle's algorithm can be replaced by simpler $O(n \log n)$ algorithms in [GJPj78] or [HM83].

It follows from the above discussion that, without preprocessing, linear time is the best achievable for polygon intersection and containment problems. However, in practice there are situations which require sub-linear query time for intersection and containment problems, as such queries will be performed a great number of times. Also, there are occasions when we want to examine all the locations where P can be placed without overlapping Q in order to select a location according to some criterion. These two requirements give rise to the demand for a preprocessing tool that can characterize all the non overlapping placements of P with respect to Q that also yield fast query time. As shall be seen in this chapter, such a demand is fulfilled by the Minkowski sum and difference.

3.1.2 Configuration Space Approach

The robot motion planning problem is the problem of finding a continuous motion to transport an object from a starting position to an ending position such that the object is free from collision with the surrounding obstacles[1]. There is a set of parameters that uniquely determines the spatial state of the moving object. Such a set of parameters includes the position and orientation of the object. It may also include the angles between the links if the object is an articulated one. When the parameters take on a set of particular values, it gives a *configuration* of the object. The *configuration space* for the object is the space of

[1] This assumes that the object in its starting and ending positions is free from collision with the obstacles.

all the possible values for the parameters. A configuration is represented as a point in the configuration space.

A configuration in which the object is free from collision with the surrounding obstacles is called a *free* configuration, otherwise it is *forbidden*. The collection of free configurations is called *free space*. And the complement of the free space in a configuration space is called *forbidden space*. It has been shown that both free space and forbidden space can be partitioned into connected components which are called *free regions* and *forbidden regions* respectively. Any path that connects two points in a free region and is totally contained in the free region represents a continuous motion of the object that is free from collision. Thus, the motion planning problem can be reduced to finding a path connecting the points representing the initial and final configurations. This is the basic idea of the ingenious *configuration space approach*[LPW79] [Can87] in robot motion planning. In this approach, the moving object shrinks to a point and the forbidden regions can be viewed as "enlarged" obstacles.

If we restrict the general motion planning problem to the problem of planning the motion of a simple polygon *translating* in a environment consisting of one simple polygonal obstacle, then the configuration space has a connected forbidden region. However, the forbidden region may have holes, and thus the free space may have multiple connected components. The case of planning the motion of a translating a simple polygon in a closed environment whose boundary can be represented as a polygon is similar: the forbidden region is connected but the free region may have multiple components. In both cases, the components of the configuration space are closely related to the Minkowski sum. We will examine the relation in detail in the next section. However, for starshaped polygons, Theorem 3.14 shows that the structure of their configuration space is particularly simple: the free space is a single connected component and so is the forbidden space.

3.2 Definitions and Properties

We present the definition of Minkowski sum and difference in set theoretic terms. The lemmas and theorems in this section are valid for discrete or continuous point sets in the Euclidean d–space.

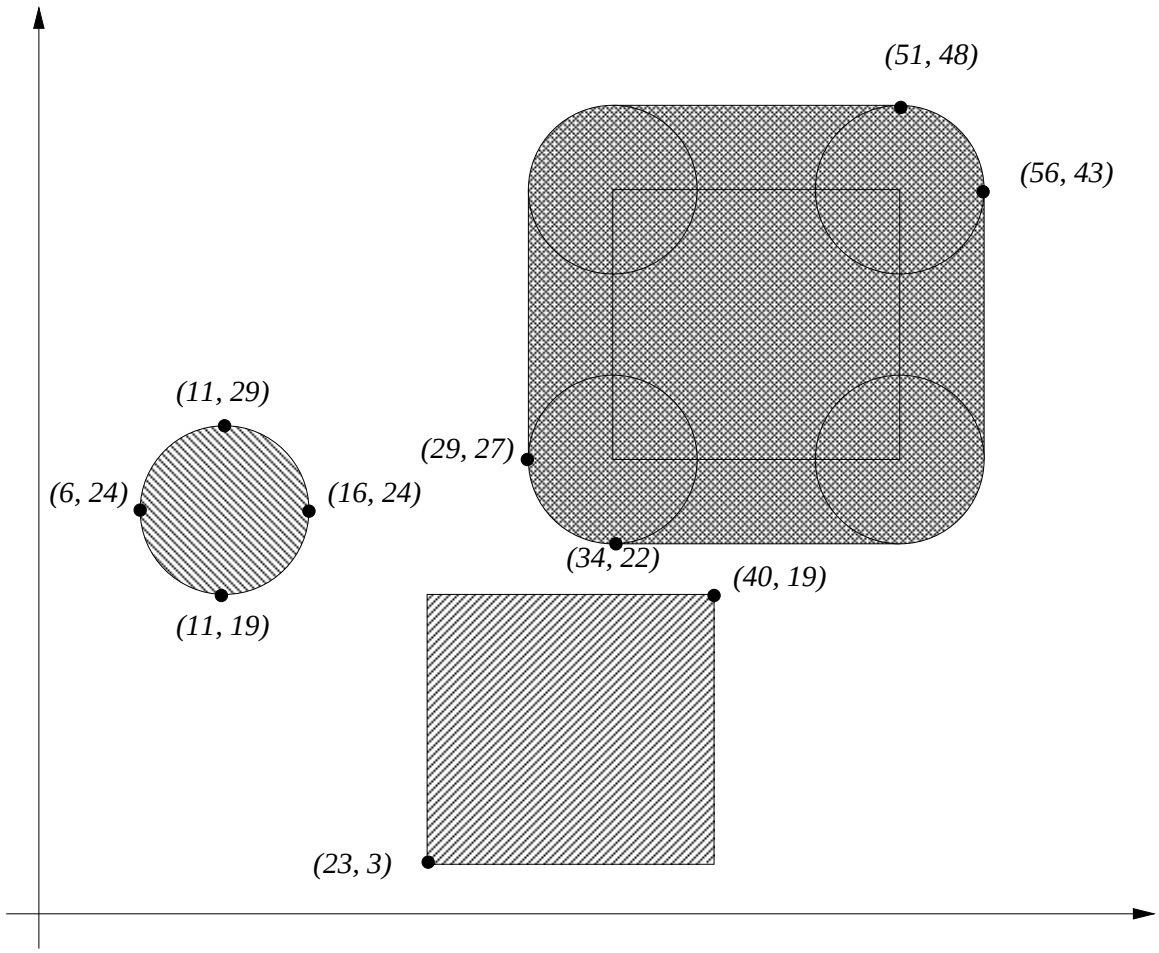


Figure 3.1: The Minkowski sum for a circle and a square.

Definition 3.1 Let A and B be two point sets in the Euclidean space, The *Minkowski Sum* of A and B, denoted by $A \oplus B$, is the point set defined as:

$$A \oplus B = \bigcup_{b \in B} A^b$$

where A^b is A translated by b:

$$A^b = \{a + b \mid a \in A\}$$

The Minkowski difference of A and B, denoted by $A \ominus B$, is a point set defined as:

$$A \ominus B = \bigcap_{b \in B} A^b$$

Example 3.1 Figure 3.1 depicts the Minkowski sum of a circle (A) with a square (B). Let C denote the Minkowski sum. This figure also provides examples of Minkowski difference:

we have $C \ominus A = B$ and $C \ominus B = A$. We would like to point out that it is not always true that $(A \oplus B) \ominus B = A$.

We first show that $A \oplus B$ can be written equivalently as an algebraic sum of A and B. This algebraic sum is sometimes easier to apply than the original definition. Hence, it can serve as an alternative definition of the Minkowski Sum .

Lemma 3.3

$$A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$$

Proof: ($\Rightarrow$) By definition, $x \in A \oplus B$ implies that there exists $b \in B$ such that $x \in A^b$. It follows that there exists $a \in A$ such that $x = a + b$. Therefore x is also an element of the set on the right hand side.

($\Leftarrow$) Similar. □

We immediately have

Corollary 3.4 $A \oplus B = B \oplus A$

Notice that, in general, $A \ominus B \neq B \ominus A$.

The next lemma establishes the relationship between Minkowski Sum and Minkowski difference.

Lemma 3.5

$$A \ominus B = \overline{\overline{A} \oplus B}$$

where $\overline{A}$ *denotes the complement of A:*

$$\overline{A} = \{\overline{a} \mid \overline{a} \notin A\}$$

Proof: By definition, we have

$$\overline{\overline{A} \oplus B} = \overline{\bigcup_{b \in B} \overline{A}^b}$$

By De Morgan's law, we have

$$\overline{\bigcup_{b \in B} \overline{A}^p} = \bigcap_{b \in B} \overline{\overline{A}^b}$$

The lemma follows from the fact that $\overline{\overline{A}^b} = A^b$. □

Symmetrically, we have

Corollary 3.6 $A \oplus B = \overline{\overline{A} \ominus B}$

The next lemma shows that the "shape" of the Minkowski sum and Minkowski difference are translation invariant.

Lemma 3.7 *Let A and B be two point sets. Let s and t be two points. Then*

$$A^s \oplus B^t = (A \oplus B)^{s+t}$$

and

$$A^s \ominus B^t = (A \ominus B)^{s+t}$$

Proof: The proof is straightforward from the definitions of Minkowski Sum and difference. □

As a consequence of Lemma 3.7, if the point sets A and B can only change location but not orientation, we need to compute their Minkowski sum or difference only once. After A and B and are placed in new locations, all we need to do is to translate the Minkowski sum or difference accordingly.

3.3 Applications: Intersection and Containment

Having studied the concept and the properties of the Minkowski Sum and difference, we now turn to their applications in intersection and containment problems and present two key theorems in this section. The theorems show that, using the Minkowski Sum and difference, intersection and containment queries on two point sets A and B can be converted into membership queries on whether a point belongs to $A \oplus (-B)$ or $A \ominus (-B)$ respectively. In the case where A and B are both polygons, the membership queries are actually point-in-polygon queries which can be answered, with proper preprocessing, in time logarithmic in the size of the polygons[2]

3.3.1 Intersection

Theorem 3.8 *Let A and B be two point sets and x be a point in the plane. Then $A \bigcap B^x \neq \emptyset$ if and only if $x \in A \oplus (-B)$, where $(-B) = \{-b \mid b \in B\}$ is the reflective image of B.*

[2]The preprocessing involves constructing the Minkowski sum or difference polygons and building search structures on the polygons. In general, the preprocessing step can be expensive.

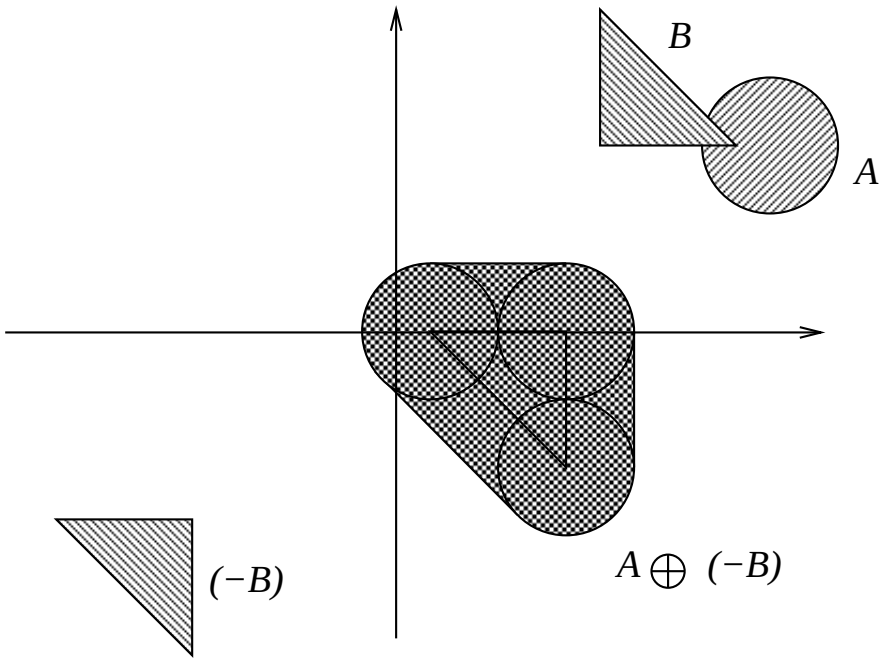


Figure 3.2: The Minkowski sum and intersection detection.

Proof: ($\Rightarrow$) Let $y \in A \bigcap B^x$. We have $y \in A$ and $y \in B^x$. That is, there exists $b \in B$ such that $y = b + x$. Therefore, we have $x = y + (-b)$. Note that $y + (-b)$ is a point in $A \oplus (-B)$. Thus, $x \in A \oplus (-B)$.
($\Leftarrow$) If $x \in A \oplus (-B)$, then we have $x = a + (-b)$ for $a \in A$ and $b \in B$. Rewrite $x = a + (-b)$ as $a = x + b$. Therefore, a belongs to both A and B^x which implies $A \bigcap B^x \neq \emptyset$ □

Corollary 3.9 *$A^s \bigcap B^t \neq \emptyset$ if and only if $t - s \in A \oplus (-B)$.*

Proof: A^s and B^t intersect if and only if they intersect after they are both translated by $(-s)$, *i.e.* if and only if $A \bigcap B^{t-s} \neq \emptyset$. □

From Corollary 3.9, we immediately obtain a useful fact which states that A and B intersect if and only if $A \oplus (-B)$ contains the origin $(0, 0)$.

Example 3.2 Figure3.2 demonstrates intersection detection for two point sets A and B. In this case, by checking the Minkowski sum of A and $(-B)$, we can verify that the intersection of A and B is non-empty since the origin is contained in the Minkowski sum.

Let A be a stationary obstacle and let B be a translating object. It is now clear that $A \oplus (-B)$ is exactly the forbidden region in the configuration space approach. The

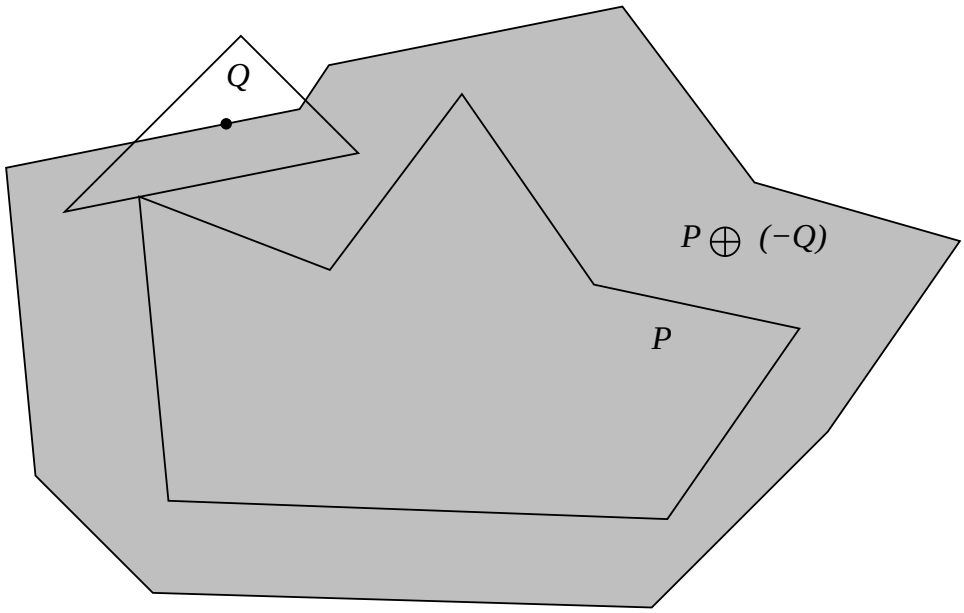


Figure 3.3: Minkowski sum and non-overlapping placement.

complement of $A \oplus (-B)$ corresponds to the *free region*. In the case where A and B are two polygons, the free region is the exterior of the polygon $A \oplus (-B)$. The free region gives all the vectors x by which B can be translated without penetrating A.

Recall that for the polygons in our marker making application, there is a local coordinate frame attached to each of the polygons. The coordinates of the points in the polygon are given in the local coordinate system. The location of a polygon in the global coordinate system is given by the global coordinates of the polygon's local origin. Let $P(p)$ denote the fact that the local origin of polygon P is placed at point p in the global coordinate system. When the location of P is not specified, we will assume P is placed at the global origin, *i.e.*, the local origin of P is coincident with the global origin.

Let P and Q both be placed at the global origin. The geometric interpretation of Theorem 3.8 for P and Q is: $Q(x)$ does not intersect P if and only if x is outside the Minkowski sum polygon $P \oplus (-Q)$

Example 3.3 Figure 3.3 shows that as long as the local origin of polygon Q is placed outside the Minkowski sum $P \oplus (-Q)$, Q cannot overlap P.

In addition to being a point set, a polygon has boundary points and interior points. We say a point x is a *boundary* point of a polygon P if for every $\epsilon > 0$, the disc of radius ϵ centered at x contains both points of P and points that do not belong to P. All the points

of P that are not boundary points are called *interior* points. If x is an interior point of P, then there exists a $\delta > 0$ such that the disc of radius δ centered at x contains only points of P.

The two polygon P and Q intersect (*i.e.* $P \cap Q \neq \emptyset$), intersection can further be classified as *overlapping*, if an interior point of one polygon is contained in the other and vice versa, or *in contact*, if P and Q are incident on their boundary points.

Now we extend Theorem 3.8 to distinguish the case where two polygons overlap.

Theorem 3.10 *Polygon P and $Q(q)$ overlap if and only if q is an interior point of $P \oplus (-Q)$.*

Proof: Since P and $Q(q)$ overlap, there exists a interior point x of P that is contained in $Q(q)$. That is, $x = y + q$ for $y \in Q$. Hence, $q = x - y \in P \oplus (-Q)$. Now, let $d_\delta(x)$ be a disc of radius δ centered at x and contained in P. Then, $d_\delta(x) \oplus (-y)$, the Minkowski sum of $d_\delta(x)$ with point $(-y)$, is a disc centered at q and is contained in $P \oplus (-Q)$. Hence, q is an interior point of $P \oplus (-Q)$.

The other direction is similar. □

Corollary 3.11 *$P(p)$ overlap $Q(q)$ if and only if $q - p$ is an interior point of $P \oplus (-Q)$.*

Proof: The proof is similar to the proof of Corollary 3.9. □

The following corollary will be used in Chapter 7.

Corollary 3.12 *$P(p)$ and $Q(q)$ are in contact if and only if $q - p$ is on the boundary of $P \oplus (-Q)$.*

Proof: Follows immediately from Corollary 3.9 and Corollary 3.11. □

3.3.2 Containment

Theorem 3.8 gives the condition for all vectors x such that B^x does not intersect A. The next theorem, which can be considered the dual of Theorem 3.8, gives the condition for all x such that B^x is completely contained in A. Before we prove the theorem we first give some intuition. Let $\overline{A}$ be the complement of A. If A is a polygon, then $\overline{A}$ is the exterior of A. For a vector x, the condition that B^x is totally contained in A is equivalent to the condition that the intersection of B^x with $\overline{A}$ is empty. By Theorem 3.8, this is the same as $x \notin \overline{A} \oplus (-B)$, or equivalently, $x \in \overline{\overline{A} \oplus (-B)}$. By Lemma 3.5, $\overline{\overline{A} \oplus (-B)}$ is exactly the Minkowski difference $A \ominus (-B)$.

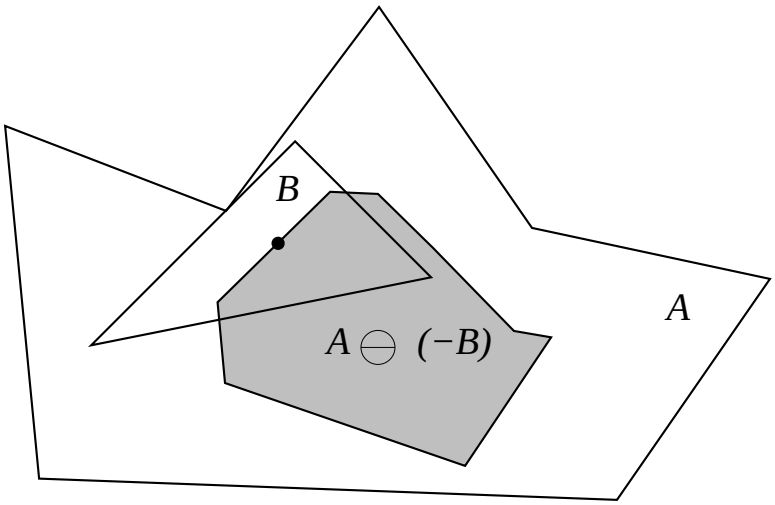


Figure 3.4: The Minkowski difference and polygon containment problem.

Theorem 3.13 *Let A and B be two point sets. Then $B^x \subseteq A$ if and only if $x \in A \ominus (-B)$*

Proof: The argument in the preceding paragraph can serve as an informal proof. Here we present a set theoretic proof.
($\Rightarrow$) $B^x \subseteq A$ implies for every $b \in B$ we have $x + b \in A$, or equivalently, $x \in A^{(-b)}$ for every $b \in B$. Hence,

$$x \in \bigcup_{b \in B} A^{(-b)}$$

Moreover

$$\bigcup_{b \in B} A^{(-b)} = \bigcup_{b \in (-B)} A^b$$

By definition, the right hand side of the above expression is exactly $A \ominus (-B)$.
($\Leftarrow$) Similar. □

Now it is clear that if we view A as a closed obstacle, the free region for planning the translation of B inside A is given by $A \ominus (-B)$. The set $A \ominus (-B)$ is also referred to as an *erosion* of A by B in image analysis [Ser82]. Taking the erosion of two point sets is an important operation in image processing for extracting the "skeleton" of an image.

Example 3.4 Figure 3.4 shows the application of the Minkowski difference in a containment problem for two polygons A and B. The shaded area represents $A \ominus (-B)$. As long as the local origin of B is placed inside $A \ominus (-B)$, B is totally contained in A.

3.4 Algorithms for Computing Minkowski Sums

Previously proposed algorithms for computing Minkowski sums have been limited to convex polygons and simple polygons, *i.e.* the polygons that are non-self-intersecting and without holes. Although their Minkowski sums can be computed efficiently, convex polygons are not suitable in our applications where the polygons are mostly non-convex. On the other hand, the algorithms for computing the Minkowski sum of two simple polygons take a long time. We have observed that starshaped polygons strike a balance between the efficiency of Minkowski sum computation and practicability. We have also observed that in our application, the majority of polygons are starshaped. The rest can be decomposed into two or three starshaped components.

3.4.1 Convex Polygons

Guibas *et. al.* [GRS83] observed that the Minkowski sum of two convex polygons can be computed in linear time by merging the edge segments of the two polygons.

3.4.2 Simple Polygons

In general, it is easy to show that an edge segment on the boundary of the Minkowski sum polygon of P and Q is part of an edge segment formed as the sum of a vertex in P and an edge in Q or *vice versa*. Let us call the edges formed by the sum of a vertex in one polygon and an edge of the other polygon candidate edges. If there are n vertices in P and m vertices in Q, then there are $O(mn)$ *candidate edges*. A natural idea for generating the Minkowski sum is to calculate the arrangement [Ede87] of the candidate edges in $O(m^2n^2 \log nm)$ time. The algorithms in [KOS91] and [AST92] for calculating the Minkowski sum of two simple polygons followed this idea. Kaul *et. al.* [KOS91] introduced the concept of vertex-edge supporting pairs which reduces the number of candidate edges. In the worst case, the Minkowski sum of two simple polygon can have $O(m^2n^2)$ edges and the same number of holes.

3.4.3 Starshaped Polygons

There is a class of polygons called *starshaped* polygons which are not as restricted as convex polygons but also easier to compute than simple polygons. A polygon P is starshaped if there exists a point k in P such that for each other point p in P, the entire segment kp

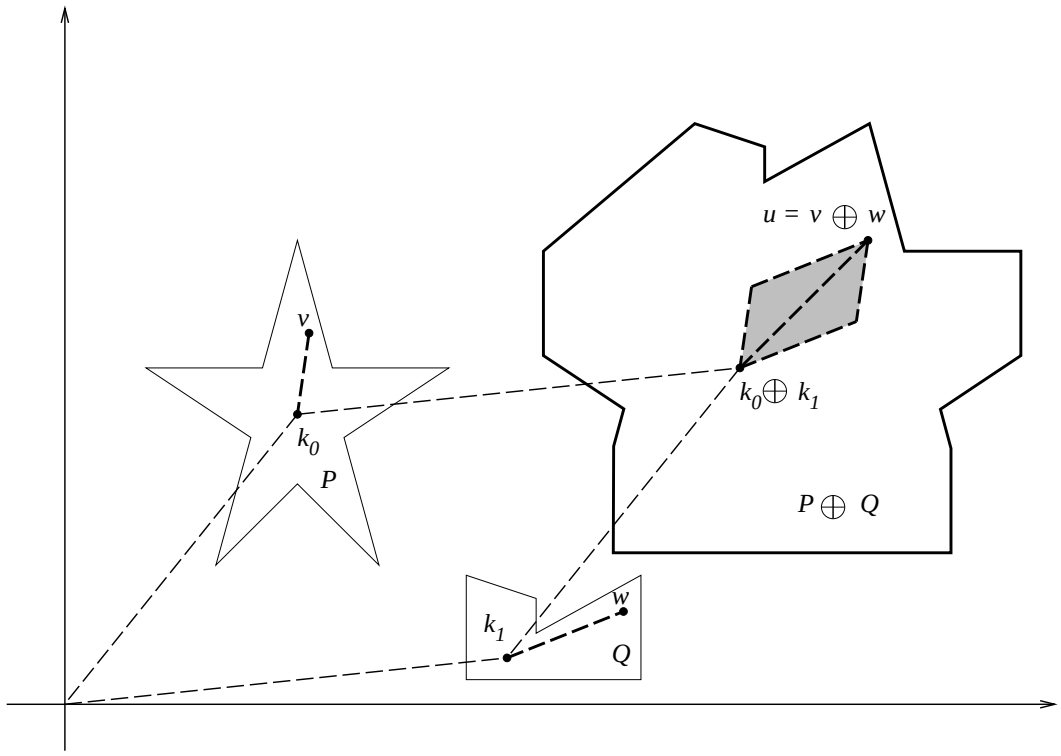


Figure 3.5: The Minkowski sum of two starshaped polygons.

lies inside P. Such a point k is called a *kernel* point of the polygon. Note that a convex polygon is a special case of a starshaped polygon in which the kernel equals the entire polygon. Polygons P and Q in Figure 3.5 are examples of starshaped polygons.

We now show how to compute the Minkowski sum of starshaped polygons. First, we prove a crucial property for the Minkowski sum of two starshaped polygons that greatly simplifies the computation of the Minkowski sum.

Theorem 3.14 *The Minkowski sum of two starshaped polygons is also a starshaped polygon.*

Proof: (See Figure 3.5) Let P and Q be two starshaped polygons. Let k_1 be a kernel point of P and k_2 be a kernel point of Q. It suffices to show the $k_0 = k_1 + k_2$ is a kernel point of $P \oplus Q$. To see this, let u be an arbitrary point in $P \oplus Q$ and by definition $u = v + w$ for $v \in P$ and $w \in Q$. Since P and Q are starshaped, we have that $k_1 v$ is totally contained in P and $k_2 w$ is totally contained in Q. Therefore, the Minkowski sum $k_1 v \oplus k_2 w$ is totally contained in $P \oplus Q$. Notice that $k_1 v \oplus k_2 w$ is a parallelogram with k_0 at one end of a diagonal and $u = v + w$ at the other diagonal. Hence, $k_0 u$ is totally contained in $P \oplus Q$. □

Remark: The theorem shows that starshaped polygons are "closed" under Minkowski sum operations. The only previously known class of polygons that is closed under Minkowski sum is convex polygons.

It follows from Theorem 3.14 that the Minkowski sum of two starshaped polygons can not have holes. Thus, the computation of Minkowski sum is reduced to calculating the outer envelope [Ede87] of the arrangement of the $O(mn)$ candidates by an angular sweepline algorithm. The outer envelope of $O(mn)$ segments can have $O(mn\alpha(mn))$ [HS86] segments where $\alpha()$ is the extremely slowly growing inverse of the Ackermann's function. For practical purposes, it can be considered a constant. The straightforward implementation of the angular sweepline algorithm runs in $O(mn\alpha(mn)\log mn)$ time. Hershberger [Her89] presented an algorithm for calculating the outer envelope of n line segments in $O(n\log n)$ time. Therefore, we have

Theorem 3.15 *The Minkowski sum of two starshaped polygons can be computed in* $O(mn\log mn)$ *time.*

Currently we are using a numerically robust implementation of angular sweepline algorithm for computing Minkowski sum of starshaped polygon. The algorithm is based on the observation that if we are unsure if a point q lies on the Minkowski sum, we can always project from the kernel point p through q and take the farthest intersection with a candidate edge. We have encountered data containing a few non-starshaped polygons but our studies have shown that all these polygons can be expressed as a union of two and very rarely three starshaped polygons. For those non-starshaped polygons, we have a decomposition algorithm to decompose the polygon into a small number of starshaped ones. During compaction, the starshaped components of a polygon are "glued" together, that is they have the same translation and tilt.

3.4.4 Monotone Polygons

A *monotone* polygon is a simple polygon that can be decomposed into two monotone chains along a direction which we call the *direction of mononicity*. A monotone chain is a piece-wise linear function with respect to the direction of monotonicity. Assume the direction of monotonicity is coincident with the y-axis, then the a monotone polygon is composed of a "upper" chain and a "lower" chain.

Like convexity and star-shapedness, the monotonicity of a polygon is a powerful tool in constraining the complexity of simple polygons and facilitates the design of more efficient

and practical algorithms.

We can consider monotonicity to be a special case of starshapedness in the sense that a monotone chain can be thought of having a kernel point in the infinity. By using essentially the same proof as the one for Theorem 3.14, we can show the following theorem.

Theorem 3.16 *If two monotone polygons P and Q have the same direction of monotonicity d, then their Minkowski Sum $P \oplus Q$ is also a monotone polygon with respect to direction d.*

The theorems shows that like the Minkowski sum of two starshaped polygons, the Minkowski sum of two monotone polygons that share the same direction of monotonicity also has no holes. Therefore, computing such a Minkowski sum is again reduced to the calculating the outer envelop. Similar to Theorem 3.15, we have

Theorem 3.17 *The Minkowski sum of two monotone polygons that have the same direction of monotonicity can be computed in $O(mn \log mn)$ time.*

To apply Theorem 3.17, we need to first identify, for two given polygons, whether there exists a direction with respect to which the two polygons are monotone. The algorithm of Preparata and Supowit [PS81] can decide whether a polygon is monotone and further find all the directions of monotonicity for the polygon in linear time. Using their algorithm, we can first calculate the ranges of the direction of monotonicity for the two polygons and do a merge in linear time to decide if the two polygons share a common direction of monotonicity.

In three dimensions, there is no known algorithm for testing the monotonicity of a polyhedron. The authors [LZ] suspect that the problem is NP-hard. However, for polygonal chains or polyhedral surface patches in three or higher dimension, testing whether there is a common direction of monotonicity can be done in linear time using linear programming.

We use three dimension as an example, the other dimensions are similar. We assume that the outward normals of a polyhedral surface have been properly labeled and are consistent (that is, the surface can be topologically transformed into a plane parallel to the $x-y$ plane and all the outward normals are upward). Let the outward normal of a face in a polyhedral surface be $\mathbf{n} = (n_x, n_y, n_z)$. The direction of monotonicity $\mathbf{d} = (d_x, d_y, d_z)$ must satisfy

$$\mathbf{n} \cdot \mathbf{d} \geq 0$$

or

$$n_x d_x + n_y d_y + n_z d_z \geq 0$$

which is a linear constraint in variables d_x, d_y, d_z.

Given several polyhedral surfaces, we build a linear constraint for each of the faces. There exists a common direction of monotonicity if and only if the linear program with the aforementioned constraints has a feasible solution. This can be tested in linear time using Megiddo's algorithm [Meg84].

If two polyhedral surfaces have a common direction of monotonicity, we can form two polyhedral terrains by projecting the two surfaces to a plane "beneath" and is parallel to their common direction of monotonicity. It is easy to show that the Minkowski sum of the two terrains so formed is also a terrain with respect to the direction of monotonicity of the two original terrains.

Chapter 4

Compaction Using a Position-Based Optimization Model

In this chapter, we describe a compaction algorithm that directly solves for the positions of the polygons without the use of time-intervals or simulation. Instead, this algorithm uses a *position-based* optimization model. The model contains artificial constraints that restrict the solution space to a convex feasible region. These artificial constraints are generated using a *locality heuristic*. The locality heuristic in turn relies on Minkowski sums of the polygons to create an explicit representation of the set of non-overlapping positions of each pair of polygons: this set is called the *free region* of the *configuration space*, or more simply, the *solution space* (see 3.1.2). The objective in the model is a linear function of the positions of the polygons. The algorithm solves this model using linear programming. The solution gives a set of new positions for the polygons. Since artificial constraints were added, the new set of positions may not be a local minimum of the objective for the original compaction problem. The algorithm computes a new model and repeats, terminating when the objective function cannot be improved by the new model. In practice, very few iterations are required to find a local minimum for the original compaction problem.

This improved algorithm overcomes the difficulties of the previous velocity-based algorithm in compacting tightly packed layouts. It runs two orders of magnitude faster than the previous algorithm.

In this chapter, we first present the position-based optimization model and then describe

the applications of the model in implementing various compaction functions.

4.1 The Theory of a Position-Based Optimization Model

4.1.1 Non-Overlapping Conditions for Two Translating Polygons

Recall that in our representation of layout problems, the position of polygon P is given by the global coordinate of the origin of its local coordinate system. As in the previous chapter, we use $P(c_P)$ to denote that P's local origin is positioned at c_P and when c_P is omitted, P's local origin is placed at the origin of the global coordinate system. We assume that the local origin of each polygon is a kernel point of that polygon. (If any polygon is not star-shaped, then we decompose it into star-shaped polygons and add the constraint that these polygons must move as one unit.) Let p and q be the new positions of P and Q respectively. From Theorem 3.1, we have that for two polygons P and Q and two arbitrary points p and q, $P(p)$ intersects $Q(q)$ if and only if $q - p$ is in $P \oplus (-Q)$. It follows that planning the simultaneous motion of P and Q is reduced to planning the motion of a single point $q - p$, which represents the relative motion of Q with respect to P, in an environment consisting of a static obstacle $P \oplus (-Q)$. Hence, to guarantee that P and Q do not overlap, we must force $q - p$ to stay in $\overline{P \oplus (-Q)}$, the exterior of $P \oplus (-Q)$.

4.1.2 A Locality Heuristic

In our applications, the exteriors of the Minkowski sum polygons are generally non-convex. In order to use linear programming, we need to find a convex subset of the exterior of a Minkowski sum polygon that can be used to formulate the non-overlapping condition as a set of linear constraints. We use a *locality heuristic* to find such a convex subset. In the rest of this section, we explain the locality heuristic in detail.

First of all, our intention is to formalize the compaction problem as a linear programming problem. Therefore, we need to express the fact that the point $q - p$ must stay in $\overline{P \oplus (-Q)}$ as a set of linear constraints on the position variable $q - p$. One way of building such a linear constraint is to specify a line segment and restrict $q - p$ to lie in the proper half-plane delimited by the line segment. For instance, let AB be a line segment and let $q - p$ be denoted by R, then from Appendix A the following cross product

$$RA \times BA \geq 0$$

specifies that R is on the right-hand side (with respect to the direction from A to B) of AB. In other words, R lies in the half-plane H_{AB} which is delimited by AB and contains right-hand side of AB. The cross product expands to the following linear constraint

$$(B_y - A_y)q_x - (B_x - A_x)q_y - (B_y - A_y)p_x + (B_x - A_x)p_y + A_xB_y - A_yB_x \geq 0 \qquad (4.1)$$

on variables p and q, where A_x, A_y, B_x, B_y are constants. We call H_{AB} the half-plane associated with the linear constraint 4.1.

Geometrically, a set of linear constraints defines a convex region which is the intersection of the half-planes associated with the constraints. Conversely, from a convex region in the plane, we can derive a unique set of linear constraints on $q - p$. Furthermore, the set of linear constraints associated with a convex region is satisfied by $q - p$ if and only if $q - p$ is contained in the convex region.

It is clear that by specifying a (non-empty) convex region in $\overline{P \oplus (-Q)}$, we can form a set of linear constraints which forces $q - p$ to stay in that convex region and thus ensures the non-overlapping condition for P and Q. Ideally, if $\overline{P \oplus (-Q)}$ is convex, we can use $\overline{P \oplus (-Q)}$ itself as the convex region to build constraints. However, $\overline{P \oplus (-Q)}$ can never be convex because it is the exterior of a simple polygon. Therefore, any convex region we choose can only be a subset of $\overline{P \oplus (-Q)}$. Hence, by forcing $q - p$ to stay in a convex subset of $\overline{P \oplus (-Q)}$ we actually restrict the freedom of the relative motion of P and Q. To reduce this limitation, we want the convex region to cover as large an area of $\overline{P \oplus (-Q)}$ as possible.

Additionally, if c_P and c_Q are the current positions of the origins of P and Q, we want the new relative position $q - p$ to be reachable from the current relative position $c_Q - c_P$ via a continuous motion of P and Q. Hence, we also require that the convex region include $c_Q - c_P$.

The purpose of the *locality heuristic* is to find a region in $\overline{P \oplus (-Q)}$ with the three requirements we just identified: convex, large, and contains the point $c_Q - c_P$. The locality heuristic finds such a convex region by first determining a point on the boundary of the Minkowski sum that is "nearest" (to be defined below) to $c_Q - c_P$. Starting from that point, it walks clockwise and counterclockwise along the boundary of the Minkowski sum. When walking clockwise (with respect to the origin of the Minkowski sum), it always makes left turns. It follows the next edge if it turns to the left (at a concave vertex of the Minkowski sum), otherwise, it extends the current edge, finds its intersection with the Minkowski sum and resumes the walk from that point. This procedure continues until the current edge can be extended to infinity. It proceeds analogously in the counterclockwise direction, making

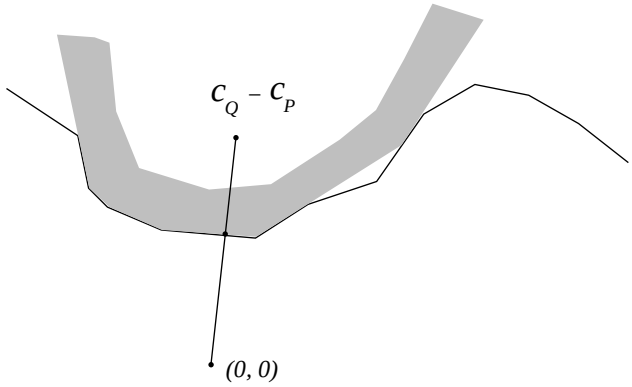


Figure 4.1: The "nearest" convex region in the exterior of the Minkowski sum.

right turns instead of left turns (see

constraints consisting of the half-planes to the "outside" of each Minkowski edge it encounters. The intersection of these half-planes is a convex subset of the feasible solution space.

Under the locality heuristic, the "nearest" point is not the boundary point with the closest Euclidean distance to $c_Q - c_P$. Instead, it is the intersection of the segment from the origin to point $c_Q - c_P$ with the boundary of the Minkowski sum. If $c_Q - c_P$ is inside the Minkowski sum, as it would be when we are separating overlapping polygons, the nearest point is obtained by extending the ray from the origin through the point $c_Q - c_P$ until it intersects the boundary. This choice of nearest point is important for the correct operation of the separation algorithm and is described more fully in Section 5.2.

Lemma 4.1 *Let P and Q be two starshaped polygons. If the origin of the Minkowski sum $P \oplus (-Q)$ is in its kernel,*[1] *then the locality heuristic terminates correctly.*

Proof: Consider the clockwise walk. The angle about the origin always increases in a clockwise direction. Since the path taken must remain visible from the starting point (and indeed from every point in the convex region), the total angle moved about the origin cannot exceed 180 degrees. Therefore the walk must terminate, and the only way it can terminate is by extending an edge to infinity. □

In the locality heuristic, if the line segment connecting $c_Q - c_P$ and the origin of the Minkowski sum passes through a convex vertex of the Minkowski sum, then either edge of that vertex can be chosen to generate the convex subset of the free space. This arbitrary

[1] If P and Q are both star-shaped, then $P \oplus (-Q)$ will be star-shaped and thus have a non-empty kernel.

choice is the same as the one made for a vertex-vertex contact in the velocity-based model (see Section 1.2.4 and Section 2.6).

4.1.3 Linear Constraints with the Boundaries of the Container

The convex subset $\mathcal{C}$ obtained by the locality heuristic naturally yields a set of linear constraints for a pair of polygons. In addition to the linear constraints for pairs of polygon, we need to take into account the constraints between the polygons and the boundaries of the container. These constraints are quite straightforward to build.

Let polygon P be adjacent to the upper boundary of the container and the let h_y be the largest y-coordinate of a vertex of P in its local coordinate system[2] Let H be the height of the container. Then, P does not overlap the upper boundary of the container in its new position p if and only if

$$p_y + h_y \leq H,$$

which is a linear constraint in p_y, Constraints with the left and lower boundaries are built similarly.

The right boundary sometimes is allowed to move. We use a variable l to represent the current length of the container. Let w_x be the largest local x-coordinates of P. The non-overlapping constraint between P and the right boundary becomes

$$p_x + w_x \leq l$$

4.1.4 The Position-based Compaction Algorithm

We first present a brief description of the algorithm and then give further explanation.

A position-based compaction algorithm.

Input "force" f_i for piece i $(i = 1, \ldots, N)$; /* *f_i's are constants* */

Assign a variable p_i for polygon i $(i = 1, \ldots, N)$;

do

L := a list of adjacent pairs of polygon returned from a sweepline algorithm;

S := $\emptyset$; /* *S is the set of linear constraints generated so far* */

foreach adjacent pair (P, Q) in L **do**

[2]Here, we assume the P is not rotated or flipped. If this is not the case, h_y should actually calculated after taking the rotation and flipping into account. Once calculated, h_y remains constant as long as P can only translate.

Compute $\overline{P \oplus (-Q)}$;

S_0 := a set of linear constraints generated for P and Q
using the locality heuristic;

S := $S \cup S_0$;

end

foreach polygon P **do**

S_1 := the non-overlapping constraints between P and
the boundaries of the container;

S := $S \cup S_1$;

end

Set up a linear program

maximize $\sum_{i=1}^{N} f_i \cdot p_i$

subject to S;

Get the set of new positions p_i $(i = 1, \ldots, N)$ by solving the linear program;

Move polygon i to p_i $(i = 1, \ldots, N)$;

until the objective function $\sum_{i=1}^{n} f_i \cdot p_i$ stops increasing.

As in the algorithm in Section 2.1, we assign a set of desired directions f_i as the "forces" applied to each polygon; these f_i's remains constant during the iterations. However, instead of assigning velocities, we use a set p_i of variables representing the positions of the polygons.

The non-overlapping constraints of the position-based model are built in the following way. We first use a sweepline algorithm to find all the adjacent pairs of polygons. For each adjacent pair, we find the set of half-plane constraints generated by the locality heuristic in the previous section.

The optimization model is set up with all the constraints from each adjacent pair and the objective function

$$\text{maximize} \quad \sum_{i=1}^{n} f_i \cdot p_i$$

This model is a linear program, and its solution is a set of positions which represent maximal motion in the desired directions satisfying the current set of constraints.

After the motions are applied, the system reaches a minimum with respect to the set of constraints generated by the locality heuristic. The algorithm may need to iterate because when the polygons are moved into their new positions, some of the convex regions generated by the locality heuristic might change. Note that in each iteration, the same linear function

is used as objective function. The value of the objective function is either increased, which triggers the next iteration, or stays the same, at which point the local minimum for the compaction problem is reached. The final position is a local minimum because the convex regions contain a feasible open ball about the current set of positions.

Furthermore, each step of the algorithm moves the polygons from one point to another in the same convex subset of the non-convex feasible (non-overlapping) solution space. Therefore, the straight line motion for each step stays within the solution space, and the total motion is a piecewise linear subset of the solution space.

Remark 4.1 Actually, manufacturers would not mind if the polygons "leap-fogged" over each other on the way to a more compact layout. The mixed integer programming generalization of this optimization model (see Chapter 8) does exactly that.

Remark 4.2 When applying the locality heuristic, we assume that the two polygons involved are both star-shaped (or at least their Minkowski sum is star-shaped). We have encountered data containing non-starshaped polygons. Our studies have shown that all these polygons can be expressed as a union of two and very rarely three starshaped polygons. For those non-starshaped polygons, we use a decomposition algorithm[3] to decompose the polygon into a small number of starshaped components. The locality heuristic is then applied to each pair of components of the two polygons.

4.1.5 Running Time and Robustness

Because the neighbor relation between pairs of polygons can be represented by a planar graph, a linear number of neighboring polygon pairs is determined by a sweepline algorithm.[4] The convex region found by the locality heuristic usually contains a small number of edges. Thus, in practice, the total number of constraints is linear in the input size. The algorithm typically runs in two to five iterations before it stops. Leftward compaction for a marker of 120 polygons, with the largest polygons having nearly 100 vertices, runs in 20 seconds on a 28 mips Sun SparcStation.

The algorithm is also numerically very robust. It handles degenerate cases, slightly overlapped inputs and (slightly) inaccurate Minkowski sums quite well. In particular, the vertices of the Minkowski sums we use have been rounded to the nearest integer lattice point.

[3] The decomposition algorithm was designed by Victor Milenkovic and implemented by Kirat Singh.

[4] Actually, without a bound on the distance a polygon can move, potentially any pair can collide and thus the graph is complete. We place a bound on the distance to ensure a planar graph.

4.2 Compaction Functions

The position-based optimization model effectively converts a coordinated motion planning problem of many translating polygons into a linear program problem. The model is general and flexible. The non-overlapping constraints among the polygons are captured by the set of the linear inequalities generated by the locality heuristic. Several compaction functions such as leftward compaction and opening up gaps can be derived from the model by simply tailoring the objective functions. By adding bounds on the positional variables to the linear program, we can impose restrictions on the motion of each individual polygon. For example, disallowing the motion of some polygons is the same as setting the upper and lower bounds of the corresponding position variables p_{ix} and p_{iy} to their current values.

In this section, we extend the meaning of compaction to include the simultaneous motion planning of all the polygons in a layout. We call the compaction in it original definition, *i.e.* shorten the length of a layout, as *leftward compaction*. We will describe several compaction functions derived from the position-based model that are useful to manual or automated marker making.

4.2.1 Leftward Compaction

For leftward compaction, the objective function is simply

$$minimize \quad l.$$

That is, the objective is to generate a motion of the polygons that maximizes the leftward motion of the "piston" polygon while allowing all the other polygons to move freely. It is not necessary for all the polygons to move to the left to minimize l. Sometimes, certain pieces need to move to the right to give a better overall length of the marker. Such motions are captured by the linear program. Figure 4.3 shows the result of leftward compaction of the marker in Figure 4.2. Notice that during leftward compaction, polygon 0 moves to the right in order to give space for polygon 11 to move to the left.

Leftward compaction is used primarily to increase the efficiency of the final layouts which can either be generated by human or by a computer. Figure 4.4 shows the result of leftward compaction on production marker shown in Figure 1.1.

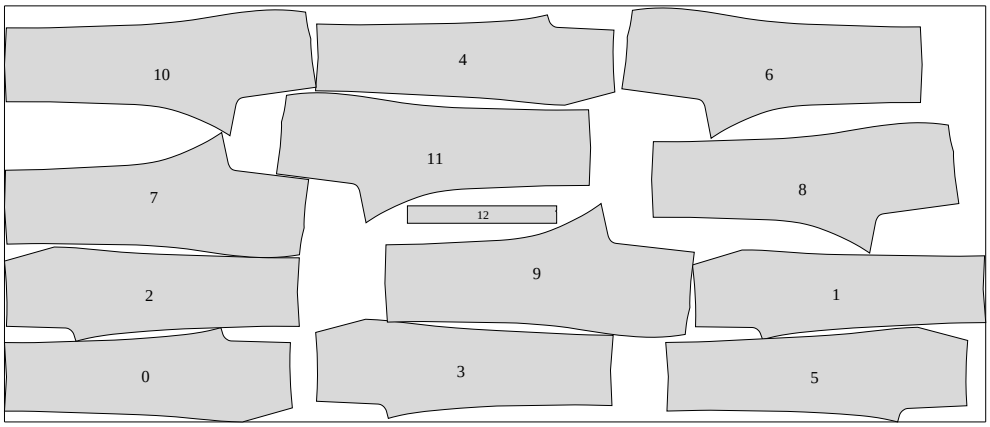


Figure 4.2: An example of leftward compaction: input.

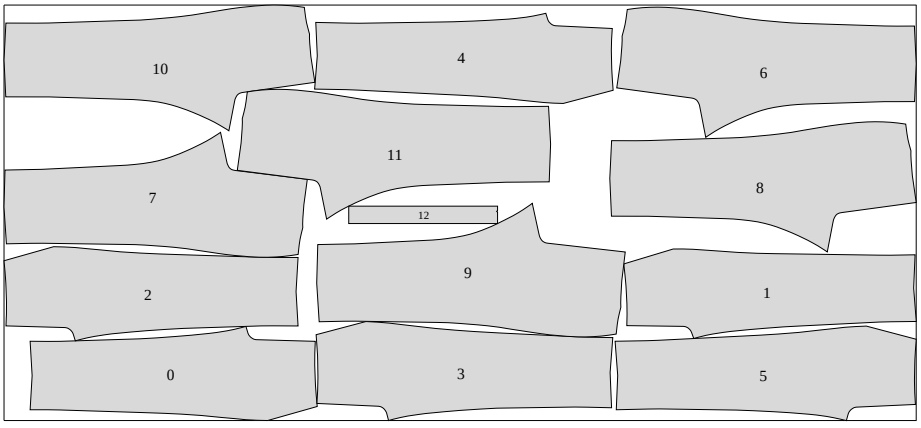


Figure 4.3: An example of leftward compaction: output.

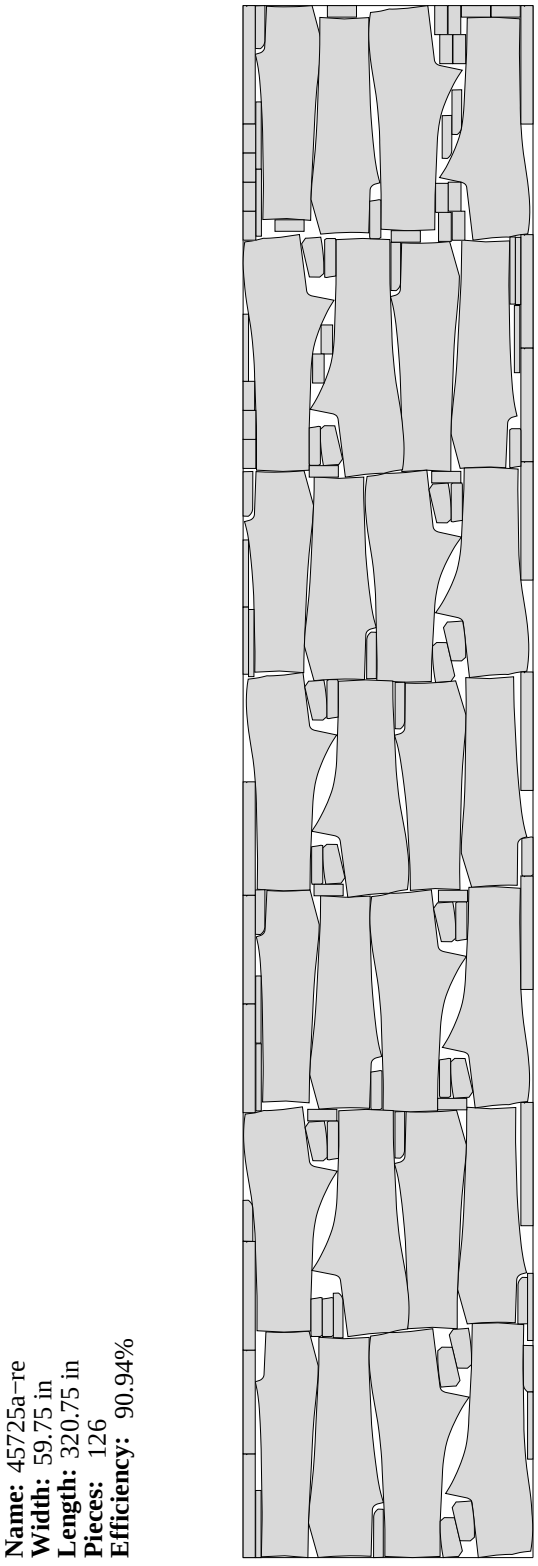


Figure 4.4: The human generated pants marker in Figure 1.1 after leftward compaction.

Improvement in Cloth Utilization

The algorithm has improved the cloth utilization of many production quality human generated markers (markers that actually go to the cutting room). The average improvement for the markers from one of the world's largest jeans manufacturers is 0.32%; the average savings in material is 1.36 inch per marker. As Section 1.2.2 indicated, a 0.1% improvement in efficiency would save about two million dollars in material for the manufacturer. In the leftward compaction example shown in Figure 4.5, the efficiency of the marker has increased by nearly 1% and total length has been shortened by 5 inches.

4.2.2 Vector Compaction

We use the term *vector compaction* to denote compaction in a fixed bounding box. It has the same objective function as in our general position based compaction model, but with the additional constraint that the right boundary is fixed:

$$l = 0.$$

The forces f_i can be assigned through the graphical user interface by drawing a vector on the corresponding pieces. Hence, the forces are referred to as *vectors*. Only the non-zero vectors need to be assigned. Usually, only a few polygons need to be assigned vectors.

Figure 4.6 shows a example of vector compaction. Polygon 2 is given a force in the upper left direction. The other polygons are moved accordingly to allow the maximum motion of polygon 2 in the specified direction. Polygon 0 is "squeezed" out from the space between polygon 2 and polygon 3. During the process, polygon 1 is pushed down while polygon 0 is not affected by the motion. This example shows, in a certain sense, that our compaction algorithm is indeed simulating the frictionless motion of rigid bodies in a viscous medium.

The most important application of vector compaction is to enlarge a gap so that more polygons can be put in it. The polygons surrounding the gap are pushed away from the gap by the outward "forces" applied on them. In manual marker making, a gap can be identified on the user interface screen. Then, a human marker maker can assign the outward vectors on the surrounding polygons manually.

Figure 4.7 depicts the effect of opening a gap. The two trim pieces in the middle cannot be fit into the gap between two large panels. "Forces" are applied to the two large panels to move one of them upward and the other downward. The result is shown in the lower part of the figure. In this example, only translation of the pieces used in compaction.

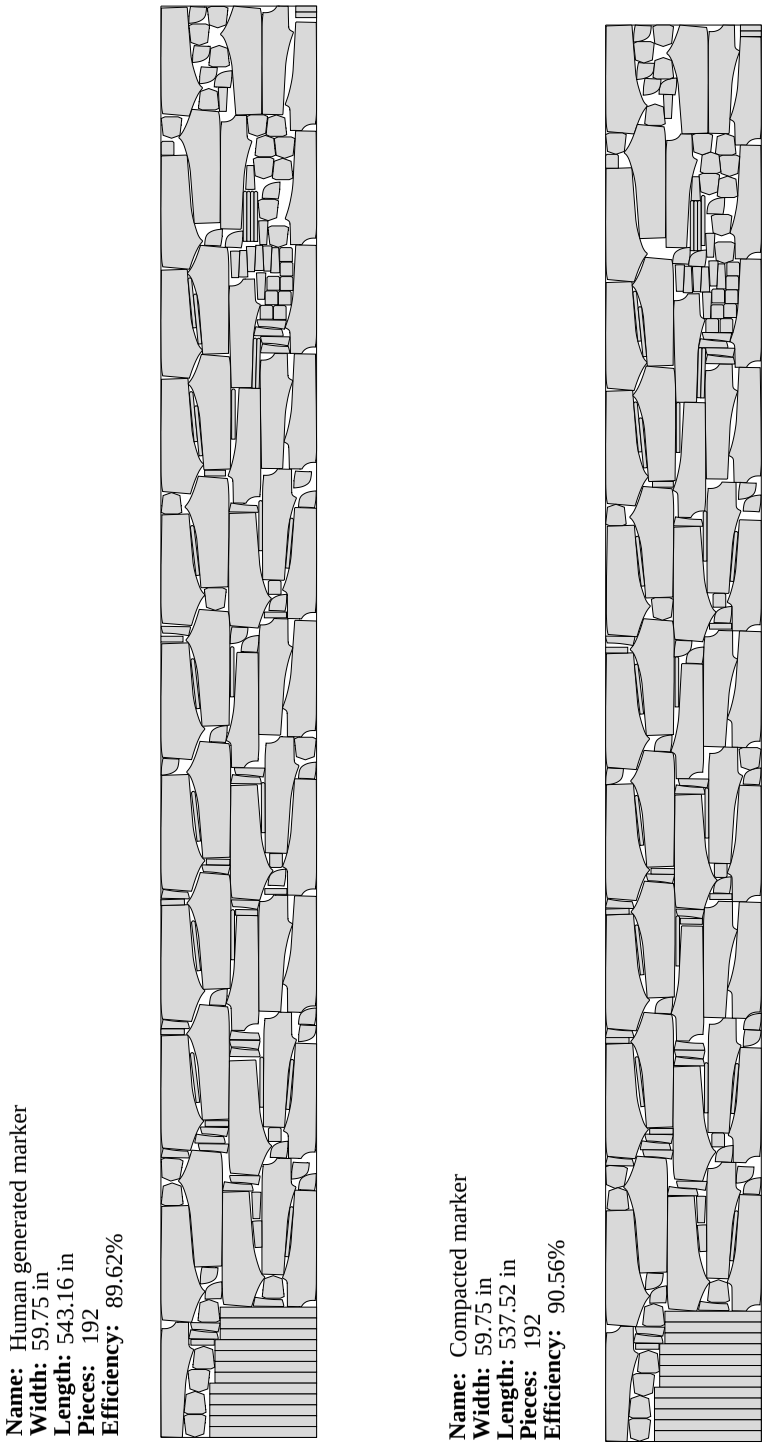
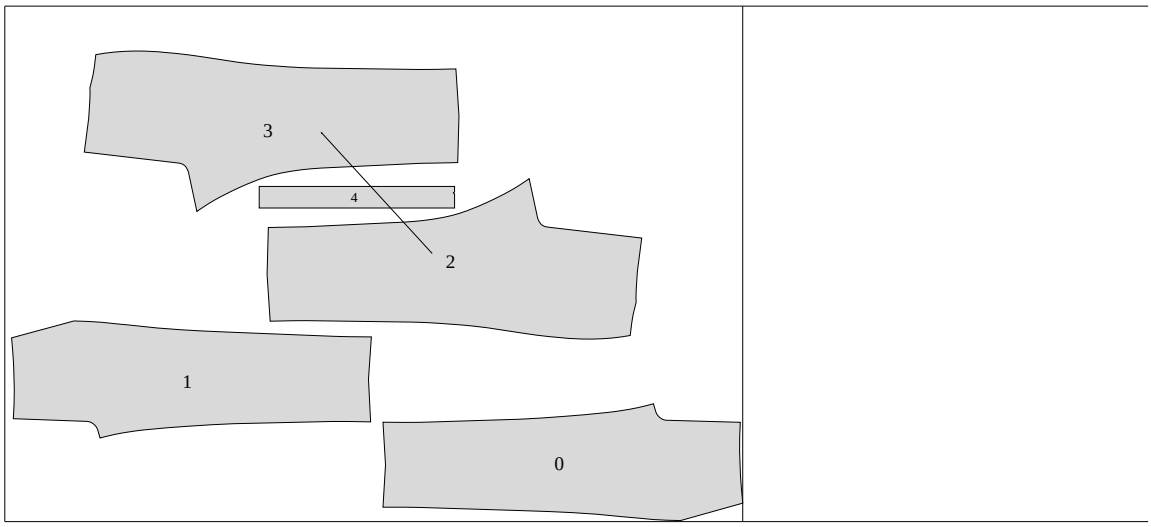
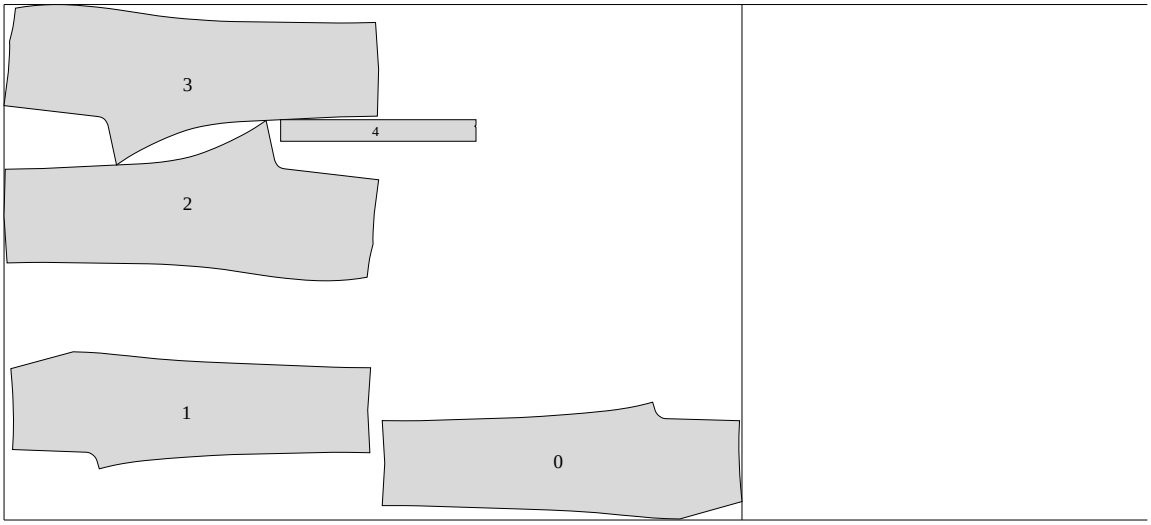


Figure 4.5: Left: A human generated pants marker. Right:The human generated marker after compaction.



(b)



(a)

Figure 4.6: An example of vector compaction.

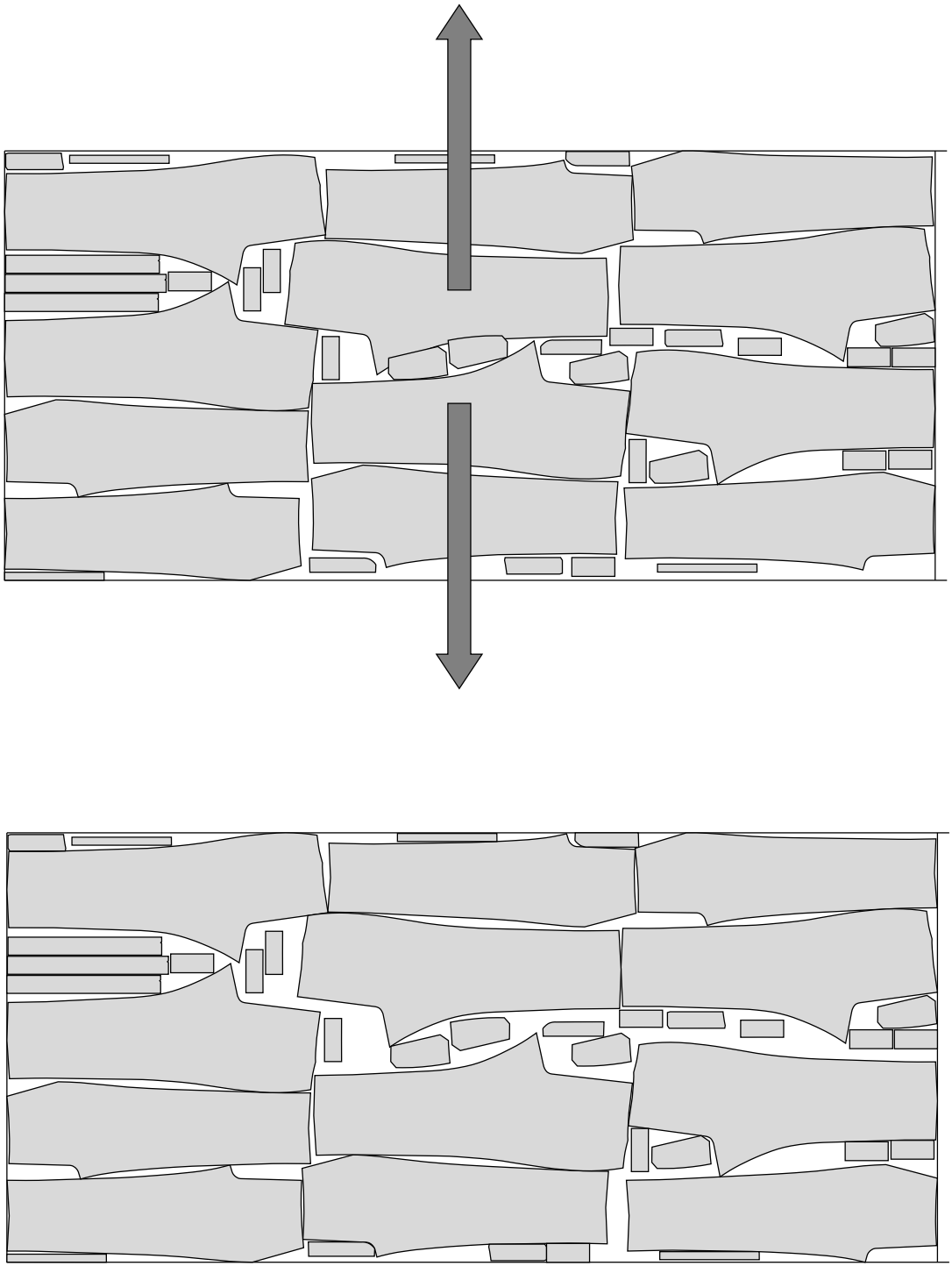


Figure 4.7: An example of opening a gap.

Opening up gaps has been an important technique used by humans to produce high efficiency markers, especially when placing the last few "difficult" pieces. In making pants marker, for example, human marker makers first place the panel pieces, and the panel placement determines to final length because humans make the utmost effort to put all the trim in the gaps amongst the panels. Frequently, the last few trim pieces can not be fitted into any of the gaps left. What human marker makers do is to choose a gap and try to enlarge it. This task is time-consuming and awkward even for experienced human marker makers when the rest of the markers has already been tightly packed. It is unlikely that they are finding the maximal size gap since they can only move one piece at a time. Hence, our compaction algorithms can greatly facilitate the manual marker making process.

In automated marker making, the gaps in a marker are identified by a program [DM94]. An automatic placement strategy chooses a set of trim polygons and a particular gap in which to place them. If the trim polygons can be placed into the gap, then vector compaction is called to enlarge the gap. An "outward" vector for a polygon surrounding that gap can be drawn from the center of the gap to the center of the polygon itself.

4.2.3 Bumping

Bumping refers to the type of compaction in which only a pre-specified polygon P_b is allowed to move within a fixed rectangle; all other polygons are fixed. The objective is to maximize the motion of P_b in a direction f_b

$$maximize \quad f_b \cdot p_b$$

Bumping has been provided by some commercial software vendors as a productivity aid in manual marker making. They use a different algorithm not based on linear programming. In traditional manual marker making, in which the pattern pieces are laid manually on a table, a marker carefully places a pattern piece so that it does not overlap any of the surrounding pattern pieces that have already been placed. Bumping releases human marker makers from the burden of calculating the non-overlapping position when placing a pattern piece. With the help of the bumping function, a human marker maker can first put a polygon in a place which is close to its desired final position, then specify a bumping vector and "bump" it into its final position. According to the author's observation on the manual marker making process of several human marker makers, bumping is involved in placing almost all the pieces.

Because commercial vendors use a naive bumping strategy, they might stop prematurely, failing to move further in a direction that can decrease the objective function. The bumping method derived from our positioned-based model will always find a local minimum of the bumping problem.

Since, all but one pieces are fixed during bumping, we can simplify our algorithm. For example, we do not need to find all the adjacent pairs; only those adjacent to P_b. Therefore, instead of a sweepline algorithm, we can use a linear time algorithm to find all the polygons adjacent to P_b. The linear program generated is small since it contains constraints for only a few pairs of polygons (P_b appears in every pair) and it has only two variables. Therefore, the algorithm runs very fast.

Bumping is an aid in several of our trim placement algorithms. For example, in a naive trim placement strategy we had experimented, a trim polygon is placed in the leftmost position in the gap that can hold it. However, in many occasions, placing the trim polygon in the leftmost position can induce fragmentation of unplaced region in a gap. Bumping is applied to reduce the fragmentation. After each trim polygon is placed, bumping is called to "bump" the polygon into upper-left or lower-left corner of the gap to help to keep the remaining region of the gap useful for additional trim polygons. In most of the examples we tried, bumping has improved the final results. The total running only increase by a few percent.

Based on the bumping algorithm, we have designed a strategy to demonstrate that one-piece-at-a-time methods can not improve the material utilization of tightly packed layouts. Our strategy is basically a simulation of a generic one-piece-at-a-time method. We use a sweepline algorithm to sort the polygons by increasing x-coordinate of their leftmost vertices. Leftward bumping is then performed on the polygons according to the sorted order. Our experiments show that this strategy does not improve material utilization most of times. This provides another evidence a coordinated motion planning approach is essential for solving compaction problem.

4.2.4 Gravity Compaction

Gravity compaction is the action of applying the same "force" to all the polygons as if the polygons are in a gravity field. For a specified force f that is to be applied to all the

polygons, gravity compaction has the objective function

$$maximize \quad \sum_{1}^{n} f \cdot p_i$$

We have observed that when applied properly, the gravity compaction function can group the otherwise fragmented unplaced regions into several large chunks. An example is depicted in Figure 4.8.

A downward force has been applied to all the polygons in Figure 4.8 and as a result the unplaced regions move up to the upper part of the marker region and combine into a few larger chunks. Larger chunks of unplaced region help increase the efficiency of trim placement since they permit more flexible of nesting among the trim polygons.

Gravity compaction can also help to "reshape" the unplaced regions inside a gap. Sometimes, the trim polygons in a gap are placed in such a way that makes it impossible to place an additional trim polygon in the gap even though the total area of unplaced region in the gap is larger than the area of the unplaced trim polygons. The application of gravity compaction changes the positions of the trim polygons already placed in the gap and the configuration of the unplaced region inside the gap. It may become possible to place an additional trim polygon in the reconfigured unplaced region. Our experiments shows that after the application of gravity, more trim polygons can be fitted into a gap. This technique was first used by Karen Daniels to improve the efficiency of some naive trim placement strategies.

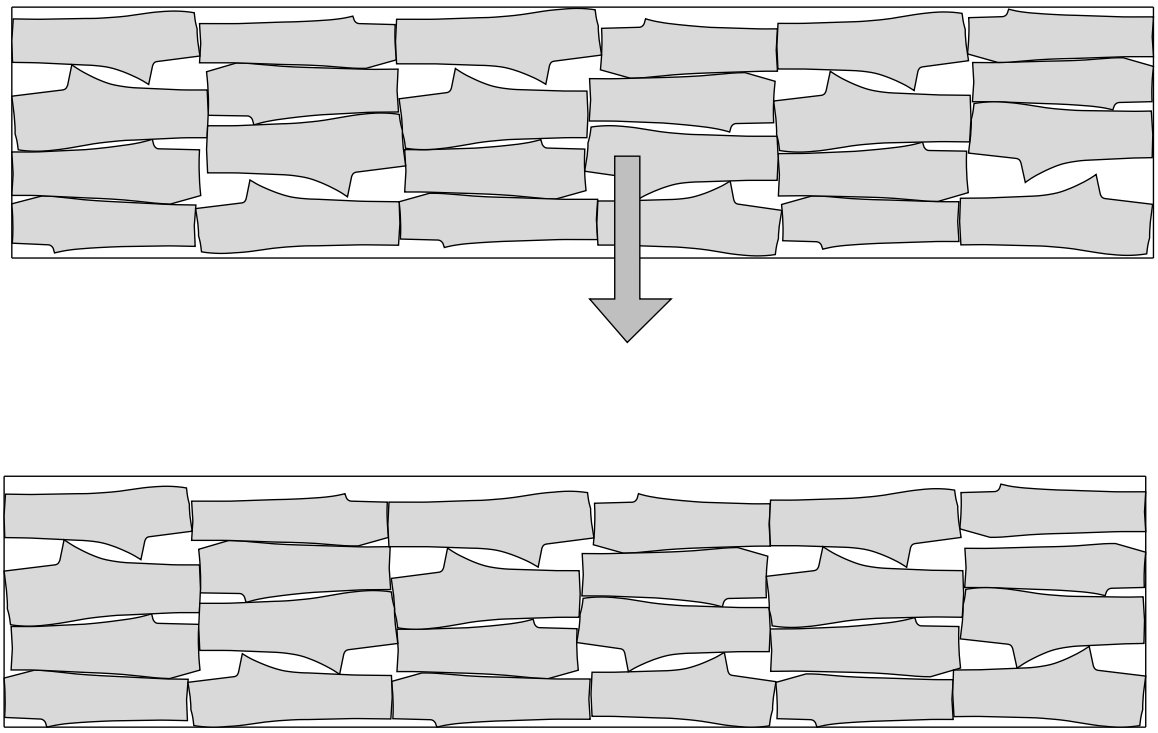


Figure 4.8: An example of gravity compaction.

Chapter 5

Separation of Overlapping Polygons and Database Driven Marker Making

In the previous chapter, we developed a position-based optimization model and used it to derive practically efficient algorithms for various compaction tasks. In marker making applications, we sometimes encounter the need for solving a seemingly different problem, the *separation* problem. The separation problem is the problem of eliminating the overlaps in a layout by moving pieces away from each other. If necessary, we would even increase the current length of the marker to give more space for resolving overlaps.

One application of an efficient algorithm for the problem is to eliminate the overlaps caused by rounding errors. A more important application is in *database driven automated marker making* systems. Such systems aim at generating markers automatically by selecting an existing marker stored in a database and then matching and substituting corresponding polygons in the existing marker. Since the substituted polygons are usually not the same as the original polygons in the existing marker, the substitution process inevitably introduces overlaps among the polygons. Whether or not the system can produce a valid marker depends on the successful elimination of the overlaps.

In this chapter, we demonstrate that the capability of resolving overlaps is embedded in the position-based optimization model. We first define and study the complexity of the separation problem. Then, we show that the same position-based optimization model also yields an algorithm which gives a locally optimal solution to the separation problem. To

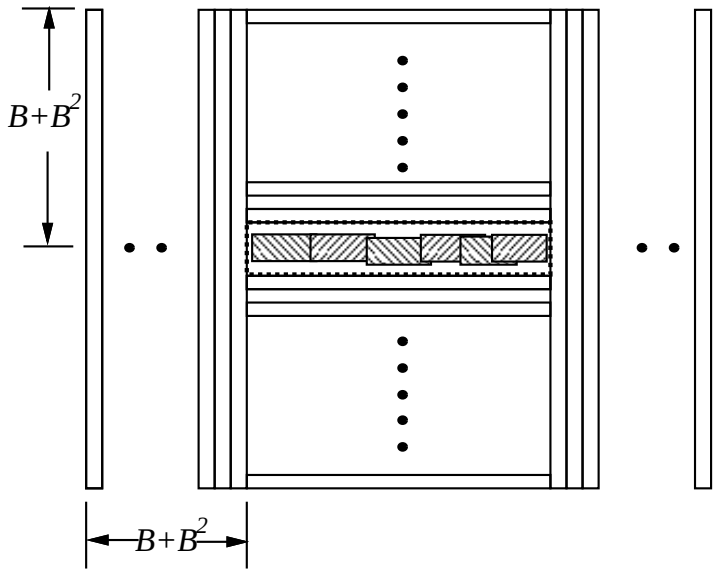


Figure 5.1: Reduction of PARTITION to separation of overlapping polygons.

our knowledge, this is the first efficient algorithm for the separation problem. Hence, it solves an industrially significant problem.

Based on this separation algorithm, we give a new scheme for database driven marker making. We use the separation algorithm together with a database of human-generated layouts to automatically generate layouts approaching human performance.

5.1 Definition and Complexity

Given a set of overlapping polygons, the *separation* problem is the problem of finding a set of translations of the polygons such that, after the translations, the polygons are in non-overlapping positions and such that the total translation of all the polygons is minimized.

First we show that finding a global minimum for the total translation is NP-complete.

Theorem 5.1 *The separation of overlapping polygons is NP-complete even for rectangles.*

Proof: Since rotation is not allowed, we can use the same argument as in [MDL91] to show that the problem is in NP. To show it is NP-hard we again reduce PARTITION to the problem. Figure 5.1 shows the construction of the reduction. Let $(a_1, a_2, \ldots, a_n)$ be the integers in an instance of PARTITION. Let $B = \sum_{i=1}^{n} a_i/2$. Place n rectangular pieces of height 1 and length a_i $(i = 1, \ldots, n)$ inside a rectangular space of size $2 \times B$

in overlapping positions. The rectangular space is then surrounded by many additional rectangular "blocks" to make sure its size is not easily enlarged. For one piece to move from one position to another position inside the rectangular space, it needs to move no more than 1 unit vertically and B units horizontally. The total motion of the pieces is therefore less than $B + B^2$. The surrounding blocks are built in such a way that to increase the length or the width of the rectangular space, they have to move a total of $B + B^2$ units distance. Therefore, PARTITION has a solution if and only if the total displacement in the separation is less than $B + B^2$. The construction can be done in polynomial time. □

In applications of the algorithm, namely the database driven automated marker making and correcting overlaps caused by rounding error, the polygons are not grossly overlapped. They tend to overlap slightly on their boundaries since the polygons being substituted are matched for similar shape and size or vertices and the position of a polygon has been rounded to the nearest integer. Next, we show that even with the additional restriction that the polygons only slightly overlap each other, the separation problem is no easier than the general case, that is, it remains NP-complete.

We define the degree of overlap r for two overlapping polygons P and Q as the minimum translation needed for P to be in a non-overlapping position with Q. We say that two polygons are slightly overlapping if their degree of overlap

$$r \leq \frac{1}{c} min\{W_P, H_P, W_Q, H_Q\}$$

for a constant $c > 1$, where W_P, H_P, W_Q, H_Q are the widths and heights of the bounding boxes of P and Q.

Theorem 5.2 *The separation of overlapping polygons is NP-complete even if the polygons just slightly overlap each other.*

Proof: We reduce the NP-hard 2-PARTITION problem to this problem. The 2-PARTITION problem is the same as PARTITION except that it requires that the two subsets each contain the same number of integers. Each piece still has height 1 but width $2Bc + a_i$. The rectangular space in the middle now has size $2 \times (2Bnc + B)$. We divide the n pieces into two rows each containing $n/2$ pieces within the middle space. The two rows do not overlap vertically. Within each row, the pieces are spread out evenly. The degree of overlap between two adjacent pieces in each row is at most

$$\frac{2B}{2Bc + a_i} \leq 1/c$$

because $2B$ is the total overlap. If there is a solution to 2-PARTITION, the moves needed to make the pieces non-overlapping are as follows: exchange two vertically adjacent pieces when necessary and do horizontal adjustment within each row. The vertical exchanges have a total displacement of at most $2n$. The total horizontal displacement within each row is at most nB because each piece has at most B horizontal displacement. So with $< 2n(B+1)$ total displacement, we can solve the separation problem. We put $2n(B+1)$ surrounding blocks on each side to restrict the movement of the pieces within the middle space. □

It is easy to see that the constant c can be replaced by a polynomial $F(B, n)$ and the proof still works.

Remark 5.1 In the proofs of Theorem 5.1 and Theorem 5.2, we assumed that the distance used is the usual Euclidean distance. We observe that the proofs are also valid using L_1 distance. The L_1 distance between the two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is defined as follows.

$$d_{L_1}(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$$

Remark 5.2 We mentioned in Section 1.2.3 that we will prove the compaction problem to be PSPACE-hard in Chapter 9. The gap in complexity between the compaction problem and the separation problem is accounted for by the fact that the compaction problem is defined as a motion planning problem in which the polygons cannot pass through (or jump over) each other, while in the separation problem there is no such restriction. We would like to point out that if we eliminate the above mentioned restriction in the compaction problem, then compaction, separation and packing all become equivalent. The NP-complete layout problem can be reduced to separation through essentially the same reduction as the on in the proof of Theorem 5.1 : In Chapter 8, we will develop a mixed integer programming (MIP) formulation of compaction. Under the MIP formulation, the polygons are permitted to pass through each other. The formulation gives a direct reduction of compaction to MIP which shows compaction is in NP. Thus, we establish the NP-completeness of compaction when the polygons can pass through each other. The same MIP formulation also gives a direct reduction of the strip packing problem to mixed integer programming. In fact, compaction and strip packing become equivalent under the formulation.

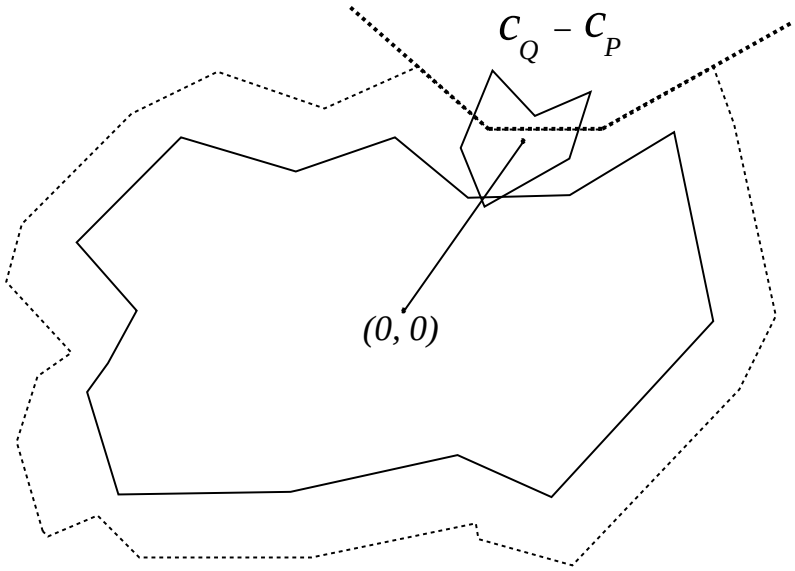


Figure 5.2: Minkowski sum of two slightly overlapped polygons.

5.2 Algorithm for Separating Overlapping Polygons

In this section, we describe a separation algorithm that finds a local minimum for the separation problem from the position-based optimization model. On the surface, it seems that the separation problem may require some different techniques. We observe that the position-based model is applicable to the separation problem since the non-overlapping conditions for adjacent polygons are prerequisites to a feasible solution of the position-based optimization model. Hence, the position-based optimization model offers a unified approach to both problems.

Let P and Q be two overlapping polygons and their current positions be denoted by c_P and c_Q. If we displace P by d_P and Q by d_Q, then the two polygons will not overlap in their new positions ($c_P + d_P$ and $c_Q + d_Q$ respectively) if and only if the point $c_Q - c_P + d_Q - d_P$ is outside $P \oplus (-Q)$. The vector from $c_Q - c_P$ to the closest boundary point on the Minkowski sum yields the shortest vector $d_Q - d_P$ that separates the two polygons. Suppose we constrain $c_Q - c_P + d_Q - d_P$ to remain within a convex subset of the exterior of $P \oplus (-Q)$ that touches the closest boundary point. If we so constrain every pair of overlapping polygons, then if a feasible solution with respect to the constraints exists, the solution will give a set of displacements that separate the polygons.

We again use the locality heuristic to find the desired convex subsets. However, we note that the "nearest" point as defined by the locality heuristic is the intersection of l with

the boundary of $P \oplus (-Q)$, where l is the line connecting $c_Q - c_P$ to the global origin. Hence, this choice of "nearest" point might not be the same as the Euclidean closest point. The advantage of this choice is that it guarantees a feasible solution if the following three conditions hold:

- the polygons are not restricted to stay within a bounding box (container);
- all polygons are starshaped;
- no two polygons have their local origins at the same global position.

To see that this is the case, we first fix the positions of the polygons in the global coordinate system and scale each of the polygons in their local coordinate systems by a scaling factor λ. We choose λ to be such that after the scaling, none of the polygons is overlapping another. Then scale up the whole layout by the same factor λ in the global coordinate system. That is, perform the following projective transformation on the global coordinate system.

$$(x, y) \longrightarrow (\lambda x, \lambda y)$$

The result is a non-overlapping layout. However, this feasibility proof depends on the fact that each "nearest" point stays at the same spot on each shrinking polygon. This holds for the given definition of "nearest" point. It does not hold for the Euclidean closest point. The first condition appears unrealistic because there is always a bounding strip for a compaction in practice. However, often a wider strip of material is available from which to cut the polygons.

As shown in Figure 5.2, if two polygons are slightly overlapping, then the difference between the polygon positions will be slightly inside the Minkowski sum. The convex region found by the locality heuristic still gives a good indication of the direction of motions to separate the two polygons. Thus, we can use the locality heuristic from the previous chapter to find the convex regions. Non-overlapping constraints are built similarly as in the position-based optimization model of the previous chapter. A feasible solution that satisfies the constraints ensures the separation of overlapping polygons.

We now specify the objective function to complete the description of the algorithm. Since the goal is to find a non-overlapping configuration with minimal amount of total displacement, the objective function is therefore

$$\textit{minimize} \quad \sum_{i=1}^{n} |d_i| \tag{5.1}$$

where $|d_i|$ is amount of displacement of polygon P_i.

As discussed in Remark 2.1, $|d_i|$ is either measured in Euclidean distance in which

$$|d_i| = \sqrt{d_{ix}^2 + d_{iy}^2}.$$

or in the L_1 metric in which

$$|d_i| = |d_{ix}| + |d_{iy}|$$

where d_{ix} and d_{iy} are x and y components of d_i.

In order to formulate the problem as a linear programing problem, we must have a linear objective function. Therefore, we use L_1 distance in the objective function 5.1. Hence, the objective function becomes

$$\textit{minimize} \quad \sum_{i=1}^{n} |d_{ix}| + |d_{iy}| \tag{5.2}$$

We use a standard technique to eliminate the absolute values in the objective function. In this technique, every variable whose absolute value appears in the objective function is replaced by two new variables. For example, the variable d_{ix} which appears in the objective function 5.2 is replaced by d_{ix}^+ and d_{ix}^-. The new variables must have positive values. That is, the following bounds are added into the linear program:

$$\begin{aligned} d_{ix}^+ &> 0 \\ d_{ix}^- &> 0 \end{aligned} \tag{5.3}$$

The variable d_{ix}^+ represents the "positive part" of d_{ix} whereas d_{ix}^- represents the "negative part". All the occurrences of d_{ix} in the constraints are replaced by

$$d_{ix}^+ - d_{ix}^-$$

The occurrence of $|d_{ix}|$ in the objective function is replaced by

$$d_{ix}^+ + d_{ix}^-$$

This technique works when the absolute values only appear in the objective function. The objective function together with condition (5.3) enforce that only one of d_{ix}^+ and d_{ix}^- has non-zero value. If in the optimal solution of the linear program d_{ix} is positive, then we have $d_{ix}^- = 0$ and $d_{ix} = d_{ix}^+$. Otherwise, we have $d_{ix}^+ = 0$ and $d_{ix} = -d_{ix}^+$. In both cases, $d_{ix}^+ + d_{ix}^-$ correctly reflects the absolute value of d_{ix}.

Now we have set up the complete linear program for the separation problem. The solution of the linear program gives a non-overlapping configuration for the polygons which

also minimizes the total displacement of the polygons measured in L_1 distance. Notice that unlike the compaction algorithms in which multiple iterations of linear program solving are needed, this separation algorithm needs only one iteration.

Remark 5.3 Preferably, we would like to minimize the total amount of displacement measured in Euclidean distance in our separation algorithm. However, we notice that in the applications of our algorithm, the primary requirement is to find a non-overlapping configuration. The consideration to minimize the total amount of displacement comes secondary. Nevertheless, the L_1 distance serves as a good approximation of the Euclidean distance in our application.

5.3 Layout Made Easy

5.3.1 Database Driven Automated Marker Making

The separation algorithm of the previous section solves an open problem in database driven systems for automated marker making. Such systems, developed by several CAD/CAM firms, are based on a database of high quality human generated markers.

Some human marker makers with 20 to 30 years of experience can generate extremely high quality markers. The markers generated by marker makers with only two or three years of experience are often 1% lower in efficiency than the markers made by the most experienced marker makers. Commercially available systems automatically generate markers at least 5% below the efficiency of the best human generated markers. It is thought that if these systems could somehow start with the top quality human generated markers as an a initial configuration, then they could generate much more efficient markers. This is the idea behind a database marker making system.

Given a set of polygons to be placed, the typical database matching system applies a set of shape similarity criteria to find the marker in the database that has the closest size and shape combinations. Each polygon to be placed in the new marker is matched to a polygon in the human generated marker. Once such a match is found, it uses a "one polygon at a time" technique to place each polygon in the new marker at the position of its matching polygon in the human marker. If the polygon overlaps previously placed polygons in the new marker, its position must be "corrected" so that it does not overlap these polygons. Once a polygon is correctly placed, its position is frozen. Without a coordinated overlap correction method, this form of "one polygon at a time" correction can grossly alter the

layout of the marker and make it harder or even impossible to place the rest of the polygons anywhere near the position of their matching pieces. In some cases the correction algorithm fails to find a non-overlapping position for the polygon.

With our separation algorithm, we can give a simpler and more reliable automatic layout scheme. To distinguish our scheme from the scheme described above, we call our scheme *substitution based automated marker making*. In our scheme, we first lay out each polygon at the corresponding position of its matching polygon in the human generated layout. This layout is created regardless of whether the polygons overlap each other. Next, we apply the separation algorithm to find a nearby feasible non-overlapping placement if one exists. If no such placement exists, we can increase the length of marker to allow such a placement and apply the leftward compaction algorithm to shorten the length.

If we have no limitation in both dimensions of the marker, we can always find a nearby non-overlapping placement (see the "scaling" argument in the previous section). However, if the width is fixed, our separation algorithm can fail to find a feasible placement. We can not expect to do better because the general problem is NP-complete.

5.3.2 Shape Matching Criteria

Recent studies in computational geometry offer some general techniques on matching polygon shapes [ABB91] [AG92]. However, these techniques do not take advantage of the properties of the polygons in a specific application area possess and thus runs slowly. One the other hand, domain specific knowledge sometimes can provide much more direct and effective heuristics for matching and substituting polygons. Take pants markers as an example, there are certain characteristics about the shape of a basic component such as a panel piece. In addition, the dimensions of the pieces are governed by the size specification, *i.e.* the panel pieces for pants of size 32×36 have basically the same dimension. Therefore, ever though the exact shape of the panel pieces differ from marker to marker, the shape similarity measure for the same-sized panel pieces is presumably very high. Hence, in practice, the size factor alone could decide underlying matching rule in a database driven marker making system.

Currently, the separation algorithm works as the engine of our proposed database driven marker making system. It does not mandate any specific shape similarity matching rule. The user of our algorithm, a CAD company in textile industry, will be instrumental in designing effective matching rules for their customers.

5.3.3 An Example

We have combined our algorithms into one separation/compaction algorithm with multiple user-selected options. This section describes an example of substitution based marker making.

Using a naive polygon matching algorithm, we match a new set of polygons to the set of polygons in a human generated marker. Figure 5.3 shows the marker generated by substituting matching polygons. Using our separation/compaction algorithm, we eliminate the overlaps in the marker and then compact leftward. The resulting marker has an efficiency of 88.89%, which is comparable to human marker makers with two or three years of experience. By moving three small polygons manually to the gap on the lower right corner of the marker and running leftward compaction again, the efficiency increases to 89.49%, only 0.65% lower than the marker generated by an expert human (Figure 5.6) for the same set of polygons. This shows that starting from a good initial configuration can greatly reduce the complexity of marker making and demonstrates the applicability of our separation algorithm for database-driven marker making.

5.3.4 Cut Planning

The database driven marker making could be the only automatic marking scheme in the next couple of years that can come very close to human performance. Its impact can be well beyond our original purpose of increasing a few fractions of a percent in material utilization. A fully automatic marker scheme that is able to produce production quality marker in real time will change the whole apparel manufacturing process. We take the cut planning in apparel manufacturing as an example to illustrate the influence of the database driven marker making.

An order that comes to an apparel manufacturer consists of the demand for different quantities of garments for several different sizes. Let there be n different sizes $S_1, S_2, \ldots, S_n$ in the order. For each size S_j $(1 \leq j \leq n)$, let demand for this size in the order be D_j. The apparel manufacturer must first calculate the materials needed to fulfill the order and then purchase the materials. The *cut planning* problem refers to the calculation of the materials needed.

The current industrial practice uses a set of generic markers $M_1, M_2, \ldots, M_m$ to represent the markers to be made. Each marker M_i contains a combination of sizes and has an efficiency e_i based on statistics on the past human performance on the particular size

combination in M_i. Let a_{ij} be the number of size S_j garments that can be cut from one M_i marker.

	S_1	S_2	...	S_n
M_1	a_{11}	a_{12}	...	a_{1n}
M_2	a_{21}	a_{22}	...	a_{2n}
...	...	...	...	...
M_m	a_{m1}	a_{m2}	...	a_{mn}
Demand	D_1	D_2	...	D_n

Assume that x_i number of markers M_i need to be cut to fulfill the order. Then the cut planning problem can be formulated as an integer programming problem.

$$maximize \quad x_i e_i$$

$$s.t.$$

$$\sum_{i=1}^{m} a_{ij} x_i \geq D_j$$
$$for \quad j = 1, \ldots, n$$

$$x_i \geq 0 \quad are\ integers$$

The marker M_i is actually made by a human marker maker if $x_i > 0$. The integer program is usually solved by a greedy algorithm that limits the number of markers need to the made.

The problem with the current approach is that the real efficiency of marker M_i cannot be taken into account when solving the integer program because the marker has not been generated yet. The real efficiency might differ significantly from historical data. Thus, cut planning process may not find the real optimal solution. On the other hand, it is often too expensive and takes to lone time to have humans generate all the markers beforehand.

With a database driven marker making system that is able to generate high quality markers in real time, we can afford to generate all the markers and obtain the real efficiency data before we solve the cut planning problem. That will enable us to have a more accurate prediction of the materials needed. This capability should reduce the inventory problem when ordering more than what is needed and avoid missed shipments because of insufficient materials for manufacturing.

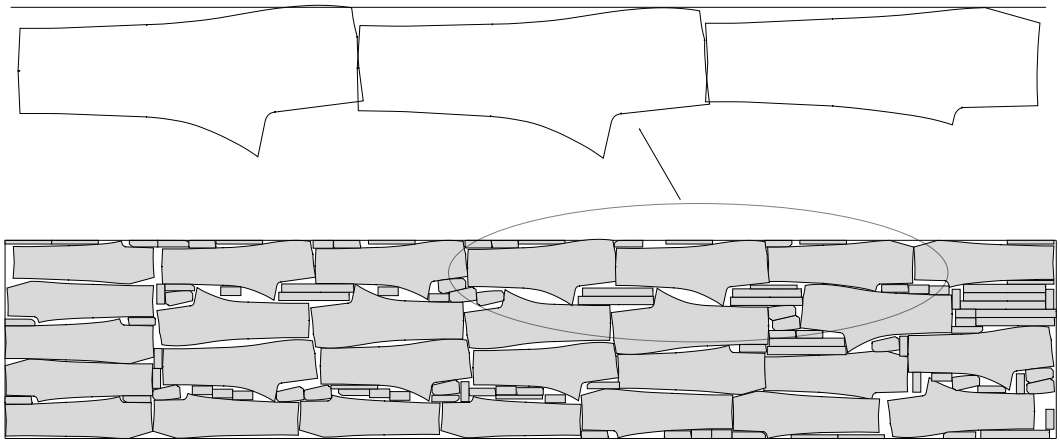


Figure 5.3: Marker generated by matching and substitution.

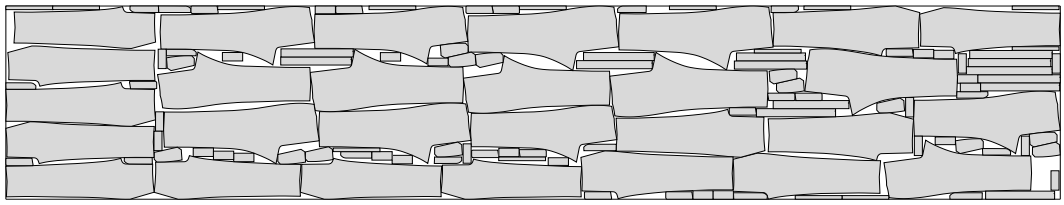


Figure 5.4: Marker after elimination of overlaps and leftward compaction. Length = 312.64 in, efficiency = 88.98%

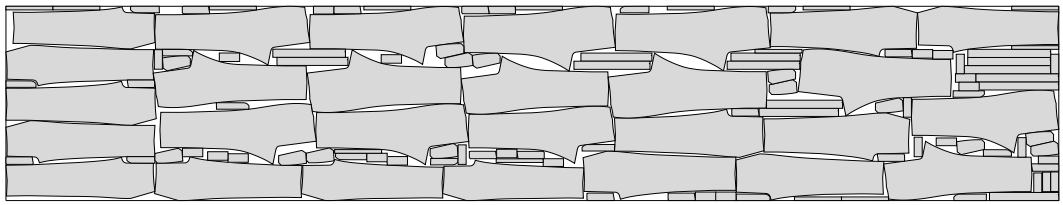


Figure 5.5: Marker after adjustment on 3 small polygons. Length = 310.87 in, efficiency = 89.48%

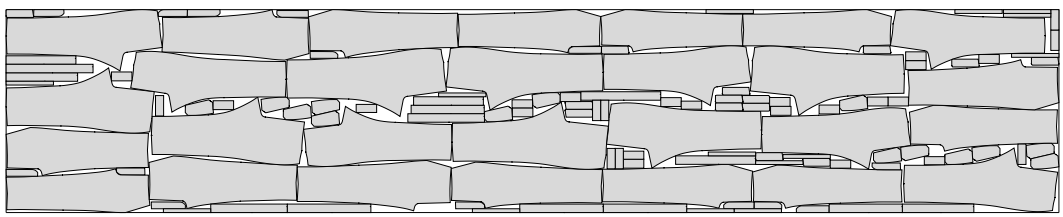


Figure 5.6: The corresponding marker generated by a human. Length = 308.61, efficiency = 90.14%

Chapter 6

Compaction with Small Rotations

In this chapter, we extend our position-based model to allow polygons to rotate by a small amount during compaction. As discussed in Chapter 1, allowing the polygons to rotate several degrees (*i.e.* tilting) can result in increased efficiency for a layout.

We present two different methods for compaction with small rotations in this chapter. The first method is based on relaxation of a polygon: finding the position and the tilt of the polygon that gives it the most freedom to move in various directions. In the second method, we consider the non-linear optimization model for the entire set of polygons involving both translations and rotations. Instead of solving this model directly, we solve a linearized version. Translation-only separation then eliminates small overlaps resulting from the linearization.

6.1 Rotational Compaction by Relaxation

6.1.1 Translational Relaxation of a Single Polygon

In this section, we focus on planning the translation of a single polygon Q in an environment consisting of k static polygonal obstacles P_1, P_2, ..., P_k that surround Q. The purpose of the motion planning is to find a point q^* and a maximal value l^* such that the center of Q can be placed at any point within a square centered at q^* and has size $2l \times 2l$. without colliding with the surrounding obstacles.

In other words, once Q's center is placed at (q_x^*, q_y^*), it will be free from collision when displaced by $(\alpha l, \beta l)$ for parameters α and β whose values are in the range $[-1, 1]$. What this means for a boundary point u of Q when Q is placed at q^* is that if we place the center

of a $2l \times 2l$ square at u then the square can not contain any points from the obstacles. If we move the center of the $2l \times 2l$ square along the boundary of Q, the square generates a "band" of obstacle-free region about the boundary of Q. We refer to this band as the *slack region* of Q. Hence, another way to look at the relaxation of Q is that if we place the center of Q at (q_x^*, q_y^*) then Q has the largest area slack region.

We call the above mentioned motion planning problem the *relaxation* problem for Q. We show next that by applying the position-based optimization model described in Chapter 4, we can find a locally optimal solution to the relaxation problem.

Recall that, using the configuration space approach, we can convert the obstacles P_i $(1 \leq i \leq k)$ into enlarged obstacles $P_i \oplus (-Q)$ $(1 \leq i \leq k)$ and reduce the problem of planning the motion for Q to the problem of planning the motion for a single point q, which represents the center of Q, in the presence of the enlarged obstacles. The free space F_Q for the motion planning of Q is the complement of the union of these enlarged obstacles.

$$F_Q = \overline{(P_1 \oplus (-Q)) \cup (P_2 \oplus (-Q)) \cup \ldots \cup (P_k \oplus (-Q))}.$$

Let us denote the square corresponding to the range $[q_x - l, q_x + l] \times [q_y - l, q_y + l]$ by $\mathcal{R}$. Clearly, solving the relaxation problem for Q is equivalent to finding largest sized square $\mathcal{R}$ contained in F_Q[1]. Using De Morgen's law, we can express F_Q alternatively as

$$F_Q = \overline{P_1 \oplus (-Q)} \cap \overline{P_2 \oplus (-Q)} \cap \ldots \cap \overline{P_k \oplus (-Q)}.$$

Hence, finding the largest $\mathcal{R}$ in F_Q is equivalent to finding the largest $\mathcal{R}$ that belongs to all of the enlarged obstacles $\overline{P_i \oplus (-Q)}$ $(i = 1, \ldots, k)$ simultaneously.

Let us represent $\mathcal{R}$ by its four corners: $(q_x + l, q_y + l)$, $(q_x - l, q_y + l)$, $(q_x - l, q_y - l)$ and $(q_x + l, q_y - l)$. Since $\mathcal{R}$ has to be contained in $\overline{P_i \oplus (-Q)}$ (for $1 \leq i \leq k$), its four corners must also be contained in $\overline{P_i \oplus (-Q)}$. We use the same locality heuristic described the last chapter to construct a large convex subset $\mathcal{C}_i$ of $\overline{P_i \oplus (-Q)}$ that contains c_Q, the current position of the center of Q. The locality heuristic is applicable here since in tightly packed layouts, such as in marker making applications, the freedom of Q to move around is very limited. This implies that $\mathcal{R}$ is usually very small and that q^* must be close c_Q. Hence, for the purpose of finding the largest $\mathcal{R}$, the convex subset $\mathcal{C}_i$ obtained by the locality heuristic gives a good approximation of $\overline{P_i \oplus (-Q)}$.

[1] More precisely, it is equivalent to finding the largest sized square in the connected component of F_Q which contains the current position of Q.

It follows from the convexity of $\mathcal{C}_i$ that if $\mathcal{C}_i$ contains all four corners of a square, then $\mathcal{C}_i$ contains the entire rectangle. Thus, to ensure that $\mathcal{C}_i$ contains $\mathcal{R}$, it is sufficient to constrain the four corners of $\mathcal{R}$ to remain in $\mathcal{C}_i$.

We use the same method as described in Section 4.1.1 to derive a set of linear constraints for each of the four corners of $\mathcal{R}$. Consider the upper right corner $(q_x + l, q_y + l)$ as an example. The linear constraint generated for this point with respect to an edge AB of $\mathcal{C}_i$ is the following.

$$(B_y - A_y)q_x - (B_x - A_x)q_y + (B_y - A_y - B_x + A_x)l + A_xB_y + A_yB_x \geq 0 \qquad (6.1)$$

which is a linear constraint on q_x, q_y and l. The constraints for other three corners with respect to edge AB are built similarly.

Next, for each obstacle P_i $(i = 1 \ldots, k)$, we calculate the convex set $\mathcal{C}_i$ from $\overline{P \oplus (-Q)}$ using the locality heuristic and built a set of linear constraints for the corners of $\mathcal{R}$ with $\mathcal{C}_i$. Let S be the collection of such linear constraints. We set up the following linear program of three variables q_x, q_y and l.

$$\begin{aligned} &\textit{maximize} && l \\ &\textit{subject to}: && S \end{aligned}$$

The solution (q_x^*, q_y^*, l^*) of the linear program gives a locally optimal solution for the relaxation of Q. That is, if the center of Q is placed at (q_x^*, q_y^*), then Q can move $(\alpha, \beta)l^*$ in various directions (α, β), where $|\alpha|, |\beta| < 1$, without colliding into the surrounding polygons. Figure 6.1 shows an example of the relaxation for piece 12.

6.1.2 Rotational Compaction of a Single Polygon

From the previous section, we see that the larger the size of $\mathcal{R}$, the larger the slack region of Q. Conceivably, the size of $\mathcal{R}$ calculated for Q with different tilts can be different. The idea behind rotational compaction using relaxation is the following: By sequentially placing each of the polygons in a position and proper tilt such that it has the largest slack region, we transform the layout into a "looser" configuration. Hence, we can subsequently apply our translational compaction algorithm to "tighten" the loosened layout.

Our first task is to find the tilt θ of Q that yields the largest $\mathcal{R}$. That is, when Q is tilted by θ, $\mathcal{R}$ as calculated by the algorithm in the previous section has the largest size. We will call this *rotational relaxation* of Q.

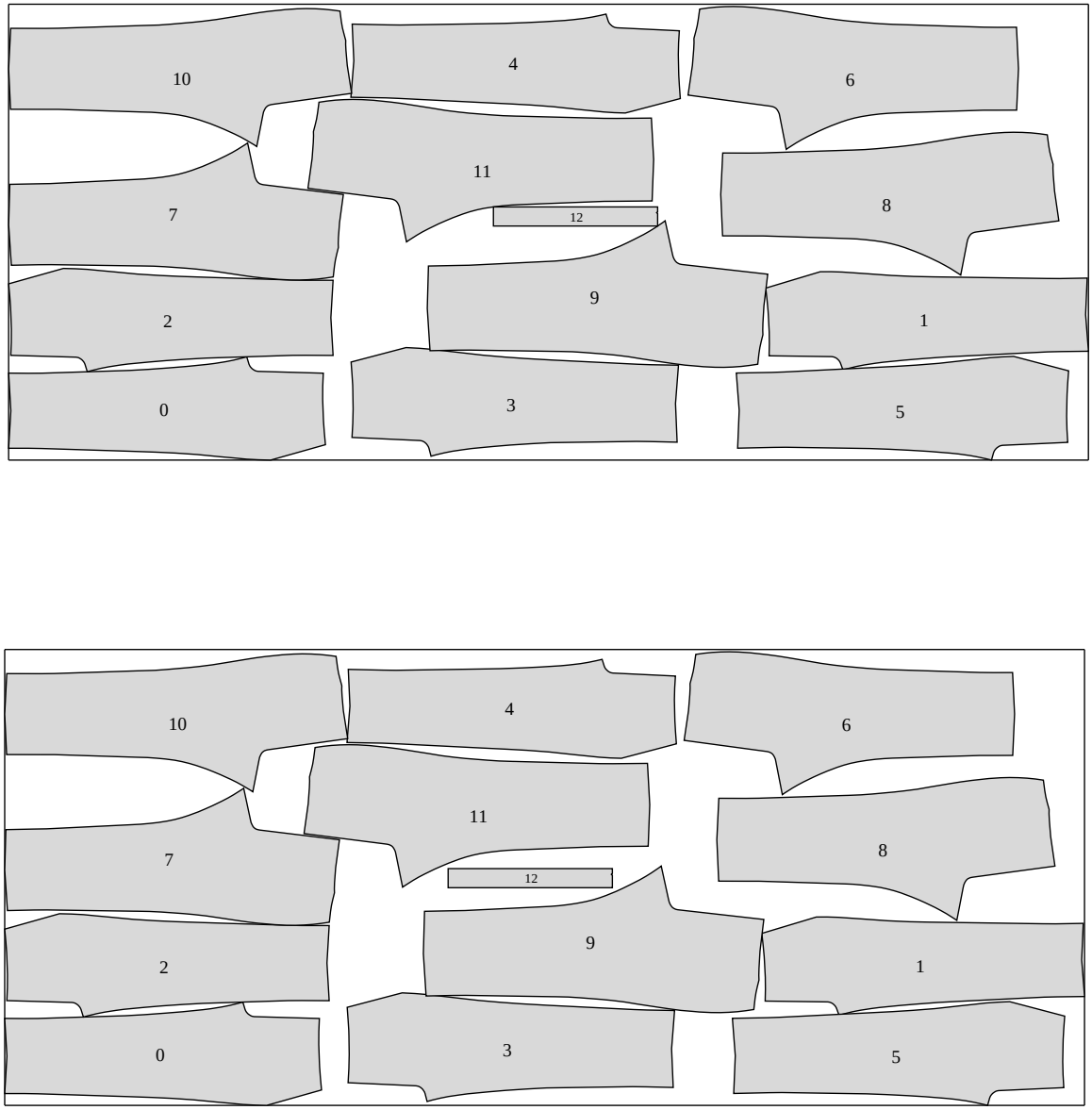


Figure 6.1: An example of translational relaxation for polygon 12.

Let the range of tilt allowed for Q be $[-\Theta, \Theta]$. Ideally, we would like to calculate the size of $\mathcal{R}$ for every angle within the tilt range and choose the largest one. However, that would require a complicated non-linear algorithm which would be difficult to make numerically stable. Therefore, we only take K equally-spaced discrete values within the range and calculate $\mathcal{R}$ for Q tilted by the rotation corresponding to a discrete value. We then set θ to the angle that yields the largest $\mathcal{R}$.

An example of rotational relaxation is depicted in Figure 6.2. Piece 12 has the position and the tilt that allow it to have the largest slack region.

6.1.3 Algorithm for Rotational Compaction Using Relaxation

The complete algorithm for rotational compaction using relaxation method is as follows.

Step 1 Do a rotational relaxation for each polygon in the layout.

Step 2 Do a translational compaction.

Step 3 Iterate Step 1 and 2 until no progress can be made.

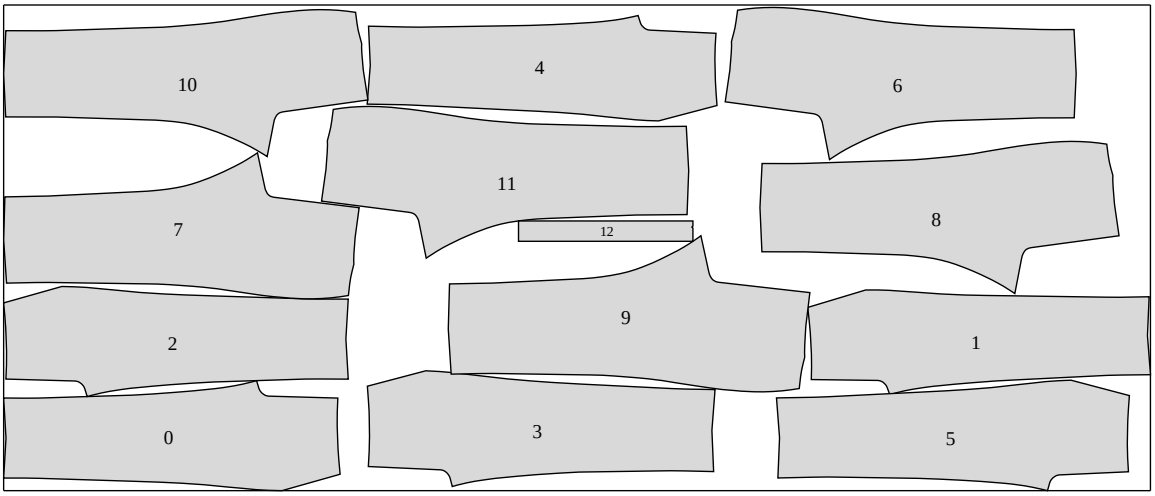
For Step 2, we can specify the same objective function used in the various compaction functions described in Chapter 4. The relaxation of Step 1 helps to "loosen" the layout and hence the translational compaction done in Step 2 usually achieves better results than without the relaxation.

Figure 6.3 through Figure 6.4 present an example of rotational compaction using relaxation on a human generated marker. Figure 6.3 shows the original marker and the result of rotational compaction using relaxation. Figure 6.4 compares the results of applying leftward translational compaction alone on the marker and applying rotational compaction using relaxation.

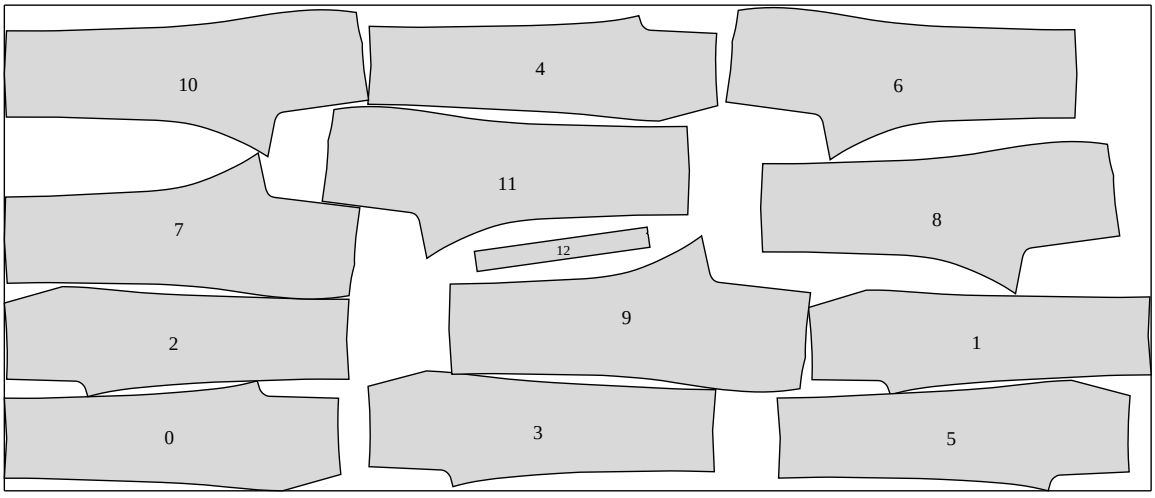
The best result we have achieved is 2.41% improvement in efficiency on a human generated production marker versus 0.79% improvement with translational compaction alone.

6.2 Rotational Compaction Using Linearization

The second method we present here for devising a practically efficient rotational compaction algorithm is *linearization*. As we have mentioned, the boundary of the configuration space of two polygons that are allowed to simultaneously translate and rotate is a nonlinear algebraic surface. To express the relation of a point to the nonlinear surface by linear



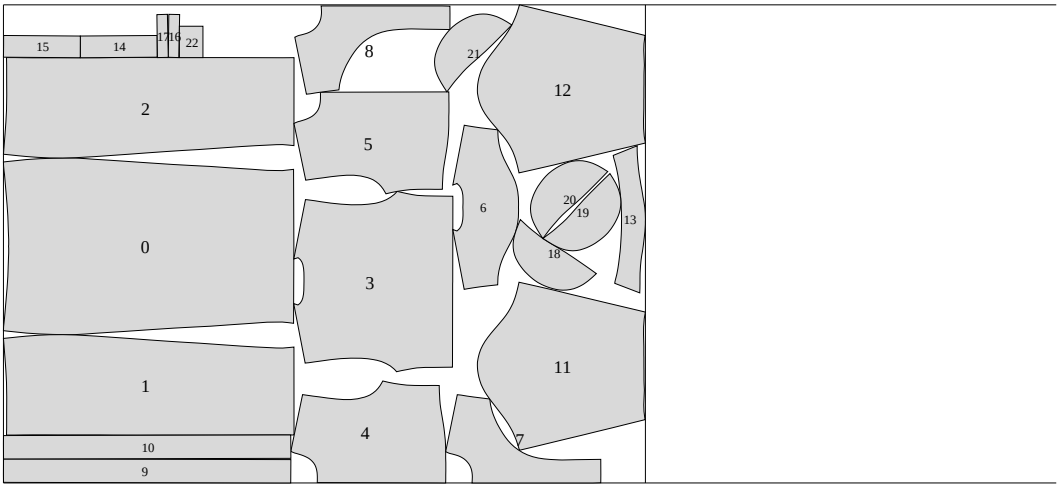
(a)



(b)

Figure 6.2: An example of rotational relaxation for polygon 12.

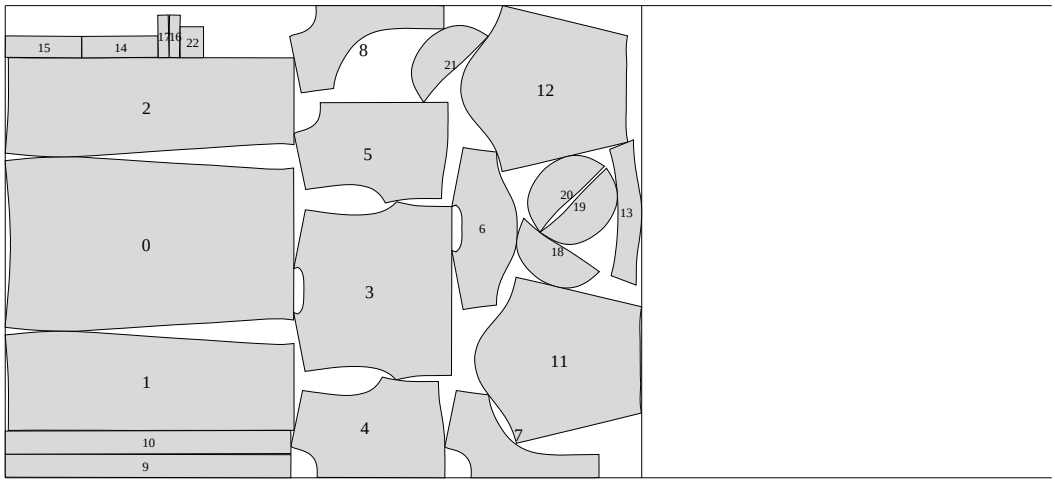
Name: 1311−5as
Width: 61.00 in
Length: 81.83 in
Pieces: 23
Efficiency: 80.85%



Name: 1311−5as−rl
Width: 61.00 in
Length: 80.25 in
Pieces: 23
Efficiency: 82.44%

Figure 6.3: An example of rotational compaction using relaxation: a human generated marker and the result rotational compaction using relaxation.

Name: 1311−5as−cl
Width: 61.00 in
Length: 81.62 in
Pieces: 23
Efficiency: 81.06%



Name: 1311−5as−rl
Width: 61.00 in
Length: 80.25 in
Pieces: 23
Efficiency: 82.44%

Figure 6.4: An example of rotational compaction using relaxation (continued): the comparison with translational only compaction on the same marker.

constraints, we cut a thin slice from the surface and try to fit it with a set of hyper-planes by linearization.

Let us consider an edge UV of the Minkowski sum $P \oplus (-Q)$ for polygons P and $-Q$, where $-Q$ is a copy of Q rotated by 180 degrees. Recall from Chapter 3 that every edge of a Minkowski sum is formed by a vertex-edge supporting pair. Let UV be formed by a supporting pair consisting of vertex A of P and edge BC of $-Q$.

Assume P is rotated around its local origin by θ_1 and $-Q$ by θ_2, where θ_1 and θ_2 are measured relative to the current tilts of the polygons. Let $|\theta_1| \leq \delta$ and $|\theta_2| \leq \delta$ for a constant δ. Assume that δ is small enough such that the corresponding vertex A' of the rotated P together with the edge $B'C'$ of the rotated $-Q$ are still an edge-vertex supporting pair which forms the corresponding edge $U'V'$ of the Minkowski sum of the rotated polygons.

Let the coordinates of A, B and C in their respective local coordinate system be (A_x, A_y), (B_x, B_y) and (C_x, C_y). The new local coordinates of these three vertices after rotation are

$$\begin{aligned} A'_x &= A_x \cos\theta_1 - A_y \sin\theta_1 \\ A'_y &= A_x \sin\theta_1 + A_y \cos\theta_1 \end{aligned}$$

$$\begin{aligned} B'_x &= B_x \cos\theta_2 - B_y \sin\theta_2 \\ B'_y &= B_x \sin\theta_2 + B_y \cos\theta_2 \end{aligned}$$

$$\begin{aligned} C'_x &= C_x \cos\theta_2 - C_y \sin\theta_2 \\ C'_y &= C_x \sin\theta_2 + C_y \cos\theta_2 \end{aligned}$$

The edge $U'V'$ on the Minkowski sum of the rotated polygons has the following coordinates ($U'V'$ is ordered counterclockwise).

$$\begin{aligned} U'_x &= A'_x + B'_x \\ &= A_x \cos\theta_1 - A_y \sin\theta_1 + B_x \cos\theta_2 - B_y \sin\theta_2 \end{aligned}$$

$$\begin{aligned} U'_y &= A'_y + B'_y \\ &= A_x \sin\theta_1 + A_y \cos\theta_1 + B_x \sin\theta_2 + B_y \cos\theta_2 \end{aligned}$$

$$\begin{aligned} V'_x &= A'_x + C'_x \\ &= A_x \cos\theta_1 - A_y \sin\theta_1 + C_x \cos\theta_2 - C_y \sin\theta_2 \end{aligned}$$

$$\begin{aligned} V'_y &= A'_y + C'_y \\ &= A_x \sin\theta_1 + A_y \cos\theta_1 + C_x \sin\theta_2 + C_y \cos\theta_2 \end{aligned}$$

In this section, we will use p and q to denote the relative displacements of P and Q with respective to their current position c_P and c_Q. Note that we change the usual meaning of p and q here for notational simplicity. The new global positions for P and Q are therefore $p + c_P$ and $q + c_Q$ respectively. We now derive the non-overlapping constraint for point $F = (q + c_Q) - (p + c_P)$ with respect to edge $U'V'$. Let us denote $c_Q - c_P$ by G. Using the same method as in Section 4.1.1, we start the derivation with

$$U'F \times U'V' \geq 0.$$

This expands to

$$(F - U') \times (V' - U') \geq 0$$

and further to

$$(F_x - F_x)(V'_y - U'_y) - (R'_y - U'_y)(V'_x - U'_x) \geq 0.$$

Substituting variables yields

$$\begin{aligned} &(G_x + q_x - p_x - (A_x \cos\theta_1 - A_y \sin\theta_1 + B_x \cos\theta_2 - B_y \sin\theta_2)) \cdot \\ &(C_x \sin\theta_2 + C_y \cos\theta_2 - (B_x \sin\theta_2 + B_y \cos\theta_2)) \\ &- (G_y + q_y - p_y - (A_x \sin\theta_1 + A_y \cos\theta_1 + B_x \sin\theta_2 + B_y \cos\theta_2)) \cdot \\ &(C_x \cos\theta_2 - C_y \sin\theta_2 - (B_x \cos\theta_2 - B_y \sin\theta_2)) \ \geq 0. \end{aligned}$$

This further expands to:

$$\begin{aligned} &(-B_y \cos\theta_2 + C_y \cos\theta_2 - B_x \sin\theta_2 + C_x \sin\theta_2) q_x \ + \\ &(B_y \cos\theta_2 - C_y \cos\theta_2 + B_x \sin\theta_2 - C_x \sin\theta_2) p_x \ + \\ &(B_x \cos\theta_2 - C_x \cos\theta_2 - B_y \sin\theta_2 + C_y \sin\theta_2) q_y \ + \\ &(-B_x \cos\theta_2 + C_x \cos\theta_2 + B_y \sin\theta_2 - C_y \sin\theta_2) p_y \ + \\ &(-B_y G_x \cos\theta_2 + C_y G_x \cos\theta_2 + B_x G_y \cos\theta_2 - C_x G_y \cos\theta_2 \\ &-A_y B_x \cos\theta_1 \cos\theta_2 + A_x B_y \cos\theta_1 \cos\theta_2 + A_y C_x \cos\theta_1 \cos\theta_2 \\ &-A_x C_y \cos\theta_1 \cos\theta_2 + B_y C_x \cos^2\theta_2 - B_x C_y \cos^2\theta_2 \\ &-A_x B_x \cos\theta_2 \sin\theta_1 - A_y B_y \cos\theta_2 \sin\theta_1 + A_x C_x \cos\theta_2 \sin\theta_1 \\ &+A_y C_y \cos\theta_2 \sin\theta_1 - B_x G_x \sin\theta_2 + C_x G_x \sin\theta_2 - B_y G_y \sin\theta_2 \end{aligned}$$

$$
\begin{aligned}
&+C_y G_y \sin\theta_2 + A_x B_x \cos\theta_1 \sin\theta_2 + A_y B_y \cos\theta_1 \sin\theta_2 \\
&-A_x C_x \cos\theta_1 \sin\theta_2 - A_y C_y \cos\theta_1 \sin\theta_2 - A_y B_x \sin\theta_1 \sin\theta_2 \\
&+A_x B_y \sin\theta_1 \sin\theta_2 + A_y C_x \sin\theta_1 \sin\theta_2 - A_x C_y \sin\theta_1 \sin\theta_2 \\
&+B_y C_x \sin^2\theta_2 - B_x C_y \sin^2\theta_2) \ \geq 0
\end{aligned}
$$

which simplifies to:

$$
\begin{aligned}
&(-B_y \cos\theta_2 + C_y \cos\theta_2 - B_x \sin\theta_2 + C_x \sin\theta_2) q_x \ + \\
&(B_y \cos\theta_2 - C_y \cos\theta_2 + B_x \sin\theta_2 - C_x \sin\theta_2) p_x \ + \\
&(B_x \cos\theta_2 - C_x \cos\theta_2 - B_y \sin\theta_2 + C_y \sin\theta_2) q_y \ + \\
&(-B_x \cos\theta_2 + C_x \cos\theta_2 + B_y \sin\theta_2 - C_y \sin\theta_2) p_y \ + \\
&(-B_y G_x + C_y G_x + B_x G_y - C_x G_y) \cos\theta_2 \ + \\
&(-B_x G_x + C_x G_x - B_y G_y + C_y G_y) \sin\theta_2 \ + \\
&(A_x B_y - A_y B_x - A_x C_y + A_y C_x) \cos(\theta_2 - \theta_1) \ + \\
&(A_x B_x + A_y B_y - A_x C_x - A_y C_y) \sin(\theta_2 - \theta_1) \ + \\
&B_y C_x - B_x C_y \ \geq 0
\end{aligned}
\tag{6.2}
$$

6.2.1 Formulation for Translation Only Compaction

As a way of verifying our derivation of the non-penetration constraint, we show that non-penetration constraint for translational compaction is a special case. Let us set the relative rotation angles θ_1 and θ_2 all to zero. We have

$$
\begin{aligned}
&(-B_y + C_y) q_x \ + \\
&(B_y - C_y) p_x \ + \\
&(B_x - C_x) q_y \ + \\
&(-B_x + C_x) p_y \ + \\
&(-B_y G_x + C_y G_x + B_x G_y - C_x G_y) \ + \\
&(A_x B_y - A_y B_x - A_x C_y + A_y C_x) \ + \\
&B_y C_x - B_x C_y \ \geq 0
\end{aligned}
$$

which is exactly the translation-only constraint we derived in Section 4.1.1.

6.2.2 Other Vertex-Edge Supporting Pair

If the vertex-edge supporting pair that forms UV came is composed of a vertex A from polygon $-Q$ and an edge BC from polygon P, then the formulation is slightly different. Let the tilts for polygon P_1 and P_2 be denoted (as above) by θ_1 and θ_2. Repeating the same derivation above, we obtain the non-penetration constraint for F against $U'V'$.

$$\begin{aligned}
&(-B_y\cos\theta_1 + C_y\cos\theta_1 - B_x\sin\theta_1 + C_x\sin\theta_1)q_x \;+\\
&(B_y\cos\theta_1 - C_y\cos\theta_1 + B_x\sin\theta_1 - C_x\sin\theta_1)p_x \;+\\
&(B_x\cos\theta_1 - C_x\cos\theta_1 - B_y\sin\theta_1 + C_y\sin\theta_1)q_y \;+\\
&(-B_x\cos\theta_1 + C_x\cos\theta_1 + B_y\sin\theta_1 - C_y\sin\theta_1)p_y \;+\\
&(-B_yG_x + C_yG_x + B_xG_y - C_xG_y)\cos\theta_1 \;+\\
&(-B_xG_x + C_xG_x - B_yG_y + C_yG_y)\sin\theta_1 \;+\\
&(A_xB_y - A_yB_x - A_xC_y + A_yC_x)\cos(\theta_1-\theta_2) \;+\\
&(A_xB_x + A_yB_y - A_xC_x - A_yC_y)\sin(\theta_1-\theta_2) \;+\\
&B_yC_x - B_xC_y \;\geq 0
\end{aligned} \tag{6.3}$$

6.2.3 Linearization

To linearize 6.2 and 6.3, we use the dominant terms in the Taylor expansion of $\sin x$ and $\cos x$ for small x

$$\begin{aligned}
\sin x &= x + O(x^3)\\
\cos x &= 1 - x^2 + O(x^4).
\end{aligned}$$

We substitute these approximations of $\sin x$ and $\cos x$ into the inequalities. Then, we eliminate the quadratic and higher degree terms and obtain the linearized form of the inequalities. The linearized form for 6.2 is

$$\begin{aligned}
&(-B_y + C_y)q_x \;+\\
&(B_y - C_y)p_x \;+\\
&(B_x - C_x)q_y \;+\\
&(-B_x + C_x)p_y \;+\\
&(-B_yG_x + C_yG_x + B_xG_y - C_xG_y) \;+\\
&(-B_xG_x + C_xG_x - B_yG_y + C_yG_y)\theta_2 \;+
\end{aligned}$$

$$(A_xB_y - A_yB_x - A_xC_y + A_yC_x) \; +$$
$$(A_xB_x + A_yB_y - A_xC_x - A_yC_y)(\theta_2 - \theta_1) \; +$$
$$B_yC_x - B_xC_y \; \geq 0$$

and for 6.3

$$(-B_y + C_y)q_x \; +$$
$$(B_y - C_y)p_x \; +$$
$$(B_x - C_x)q_y \; +$$
$$(-B_x + C_x)p_y \; +$$
$$(-B_yG_x + C_yG_x + B_xG_y - C_xG_y) \; +$$
$$(-B_xG_x + C_xG_x - B_yG_y + C_yG_y)\theta_1 \; +$$
$$(A_xB_y - A_yB_x - A_xC_y + A_yC_x) \; +$$
$$(A_xB_x + A_yB_y - A_xC_x - A_yC_y)(\theta_1 - \theta_2) \; +$$
$$B_yC_x - B_xC_y \; \geq 0$$

6.2.4 The Algorithm

The algorithm proceeds similar as the algorithm in Section 4.1.4. We first generate the pairs of adjacent polygons using a sweepline algorithm. For each adjacent pair of polygons P and Q, we compute the Minkowski $P \oplus (-Q)$ and identify a convex subset $\mathcal{C}$ in $\overline{P \oplus (-Q)}$ using the locality heuristic. For each edge of $P \oplus (-Q)$ that is included in $\mathcal{C}$, we identify the vertex-edge supporting pairs that formed this edge.[2] The non-overlapping constraints are generated using these vertex-edge supporting pairs according to the formulation in Section 6.2.3. The algorithm then solves the resulting linear program.

What is left to specify are the bounds on the rotation angles. For each polygon, we allow its tilt to change continuously within a small range. To limit the approximation error, we choose the range to be $[\theta_0 - 0.5, \theta_0 + 0.5]$ degrees. This range must be contained in the tilt limits. Hence, the final range we use is

$$[\theta_0 - 0.5, \theta_0 + 0.5] \cap [-\Theta, \Theta]$$

It is conceivable that due to the approximation of $\sin x$ and $\cos x$, overlap among the pieces may occur after each linearized rotational compaction. However, the overlaps are

[2] An edge in the Minkowski sum can be formed by more than one vertex-edge supporting pairs.

very small, and we can effectively eliminate them using the algorithm in Chapter 5. We run a overlap resolution between every two linearized rotational compaction steps to avoid accumulation of error. The complete algorithm is therefore to iterate a linearized rotational compaction step and an overlap resolution step until no further improvement to the efficiency can be made.

6.2.5 Examples

Figure 6.5 illustrates one iteration of linearized rotational compaction on a simple layout with only two polygons. The figure on the top shows the initial layout. The figure in the middle depicts the position of the polygons immediately after linearization. One overlap, which occurs between the two polygons, is magnified in the large circle. The bottom figure shows the result after the elimination of the overlaps in the middle figure. Figure 6.6 shows the final result of rotational compaction on the layout in Figure 6.5 after several iterations which bring the tilts of polygons to their limits.

Figure 6.7 shows the result of rotational compaction using linearization on a human generated marker. Figure 6.8 compares the results of rotational compaction using linearization with translational only compaction.

6.3 Comparison of the Two Rotational Compaction Methods

One would expect the linearization method to perform better than the relaxation method (*i.e.* reach a lower "energy" status for the same layout using the same objective function) based on the following two reasons. First, in the relaxation method, the rotation angle of a polygon can only be taken from several discrete values equally spaced within a range. In the linearization method, however, the rotation angle of the polygon can change continuously within the same range (through successive linearized compaction steps). Hence, the linearization method can reach the same set of position/rotation values that results in a local minimum for the relaxation method. Second, a layout can be "locked" if only a one-piece-at-a-time motion is permitted as in the relaxation method. However, the same layout may not be "locked" if simultaneous motion of all the polygons is permitted.

In practice, however, the linearization method does not outperform the relaxation method on every marker. There exist markers that give better compaction result using the relax-

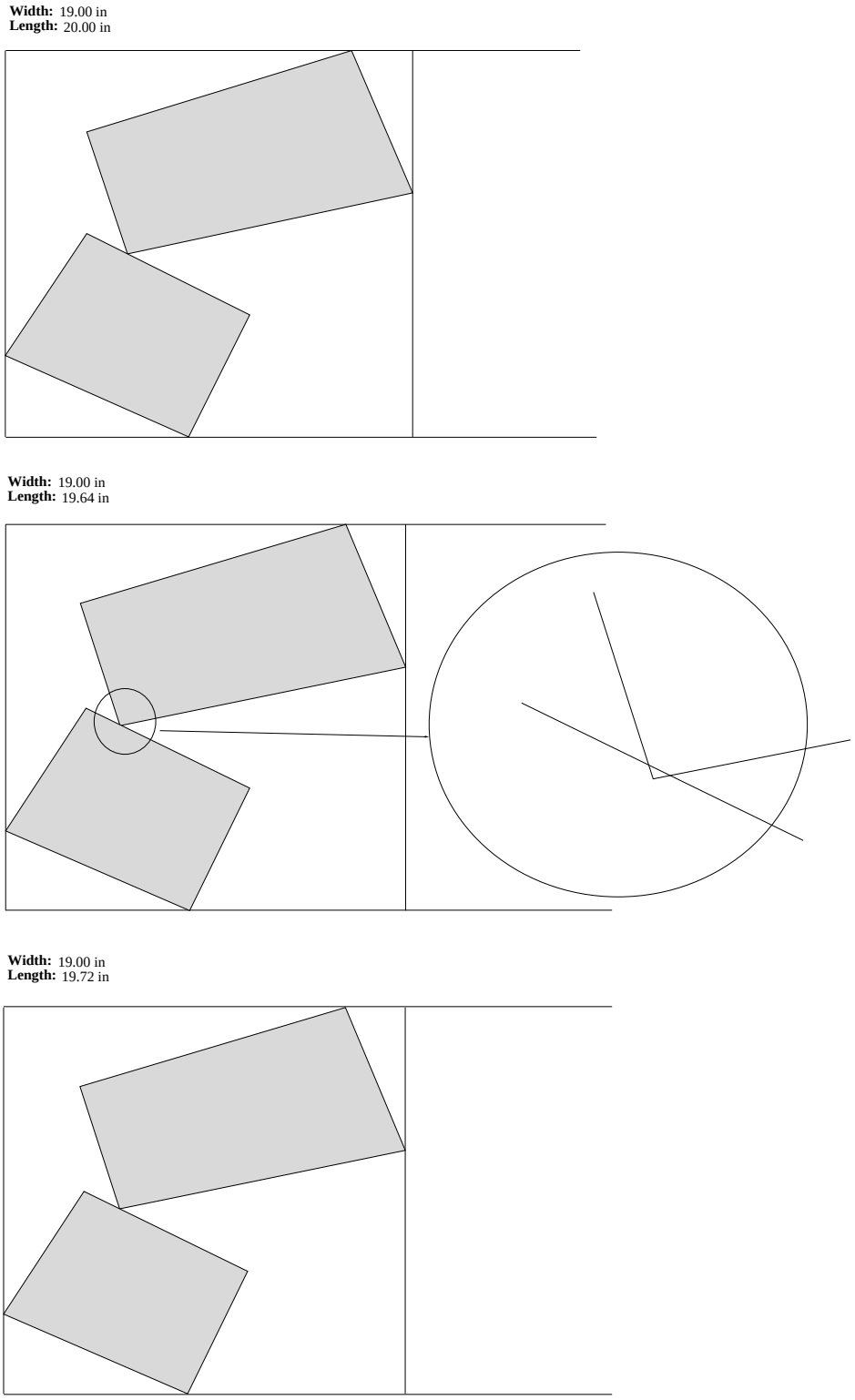
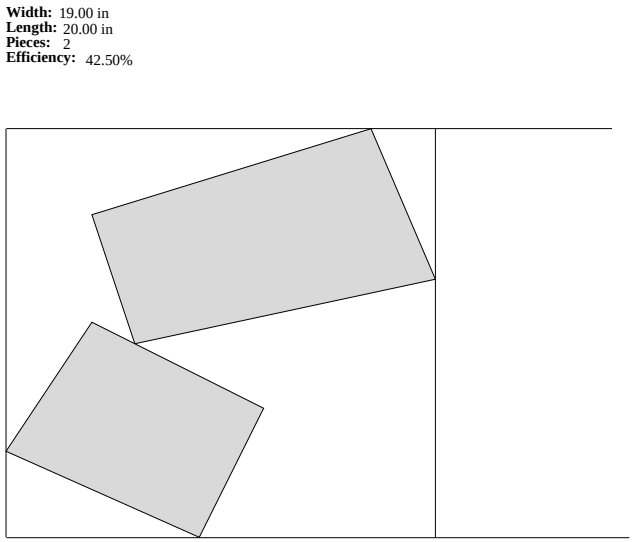
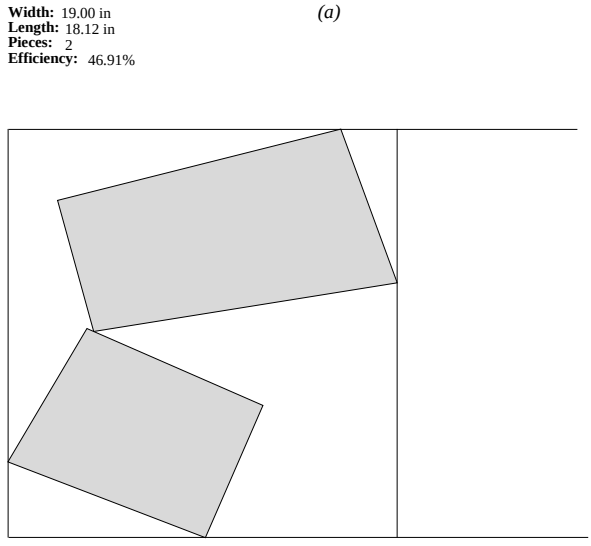


Figure 6.5: A single iteration of rotational compaction using linearization.



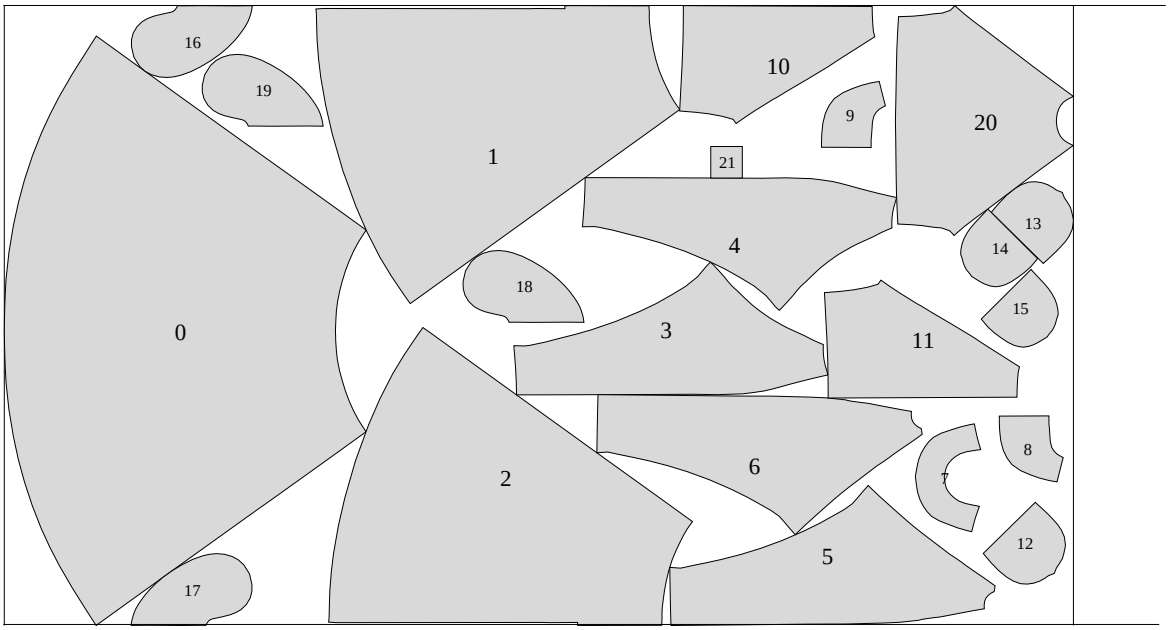
(a)



(b)

Figure 6.6: The final result of rotational compaction using linearization after successive iterations.

Name: 24412−7
Width: 59.00 in
Length: 101.74 in
Pieces: 22
Efficiency: 74.22%



Name: 24412−7−rot
Width: 59.00 in
Length: 98.42 in
Pieces: 22
Efficiency: 76.72%

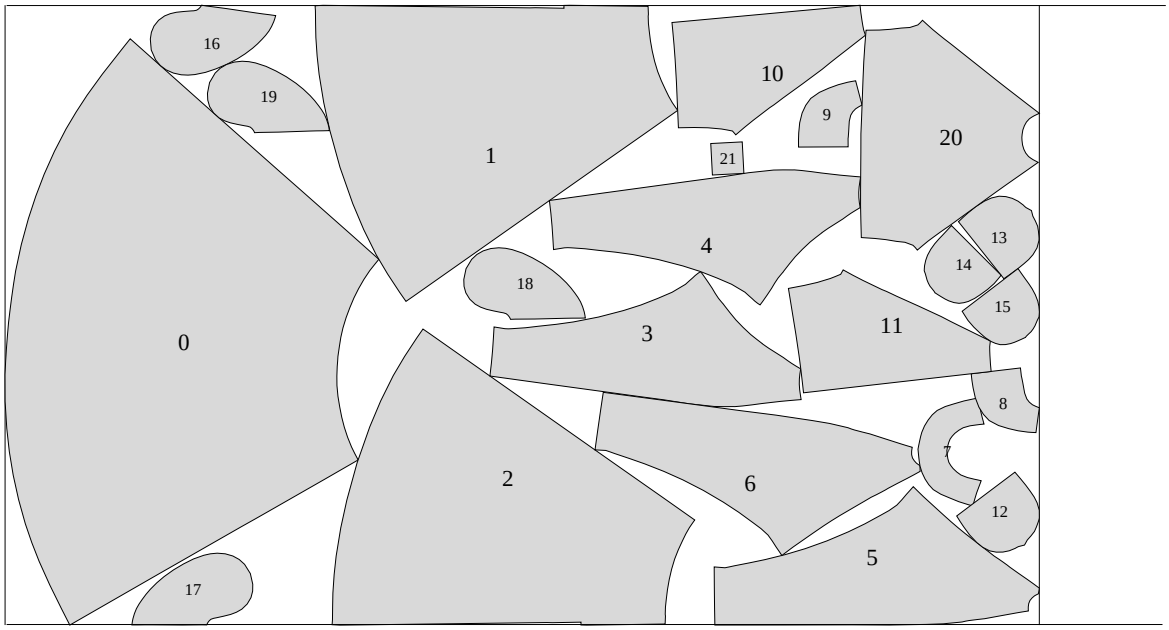
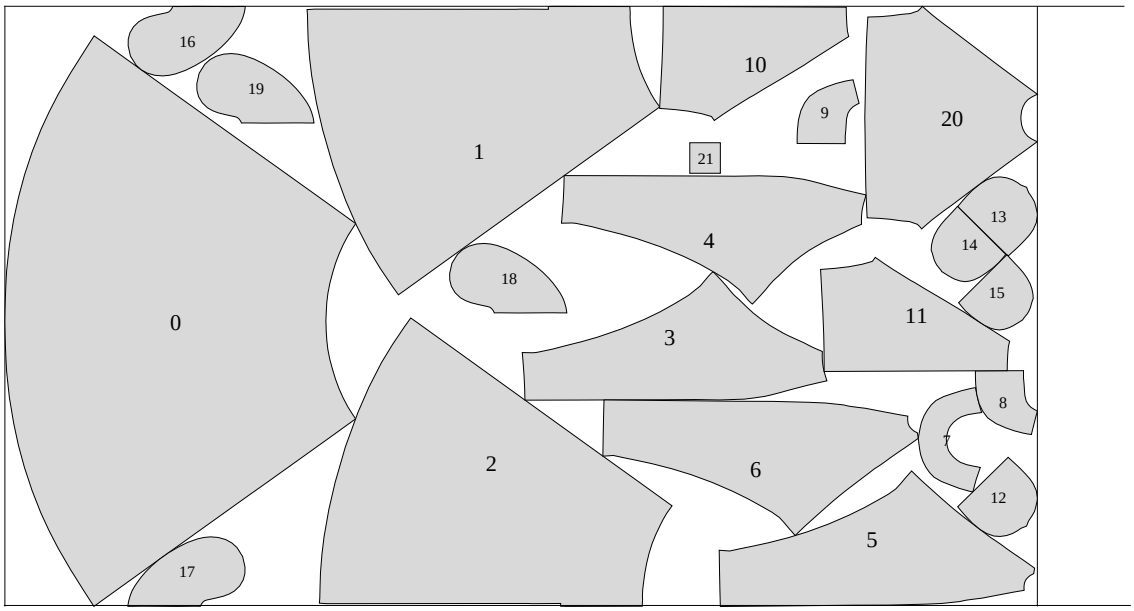


Figure 6.7: An example of rotational compaction using linearization: a human generated marker and the result of rotational compaction using linearization.

Name: 24412−7−cl
Width: 59.00 in
Length: 101.35 in
Pieces: 22
Efficiency: 74.51%



Name: 24412−7−rot
Width: 59.00 in
Length: 98.42 in
Pieces: 22
Efficiency: 76.72%

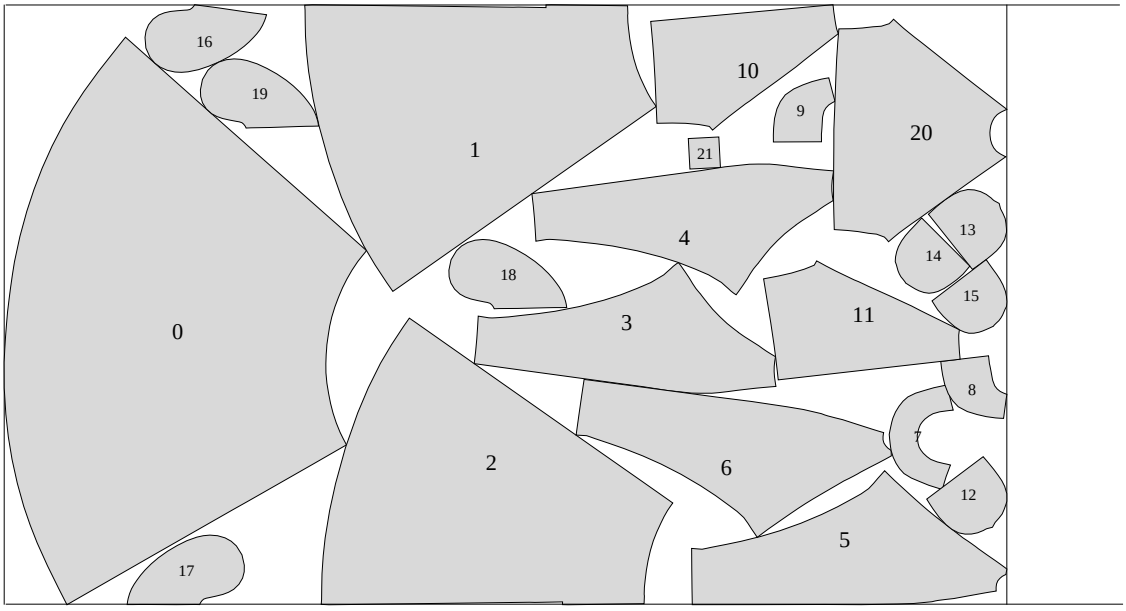


Figure 6.8: An example of rotational compaction using linearization (continued): the comparison with translational only compaction on the same marker.

ation method. This fact illustrates that locked markers for rotational compaction are very rare, so relaxation works most of the time. It also illustrates that the relaxation method can jump to more global optimum because it considers the entire valid tilt interval whereas the linearization method falls to the nearest optimum.

On the "unlocked" markers, the linearization method runs much faster than the relaxation method. This is because the relaxation method takes a relaxation step for every polygon and each relaxation step involves several iterations of setting up and solving a linear program. On the other hand, the linearization method is much slower than translation-only compaction since the small interval required by linearization makes many more iterations of linearized rotational compaction in order to converge to a local optimum. Our experiments shows that linearization method usually is about four times slower than translation-only compaction.

Chapter 7

Floating

In this chapter, we present another extension to the position-based optimization model of Chapter 4. The extended model solves the *floating* problem – the problem of distributing free space in a layout evenly among the polygons. Another way to describe floating is that it tries to increase distances between neighboring polygons as a electric field does to a set of mutually repulsive charged objects in a container. Figure 7.1 shows an example of floating.

The motivation for floating is that, in the cutting process, the cutting tool needs a certain amount of clearance in order to avoid cutting into adjacent polygons. One way of creating the clearance is to add buffering area along the boundary of a polygon. However, adding buffering area decreases the efficiency of the layout. In marker making, the primary goal is always to get maximal efficiency. Separating pieces that are touching is a secondary goal. In achieving this secondary goal, it is never allowed to decrease the efficiency. Hence, adding buffering is not viable alternative. In this chapter, we show that by using floating, however, one is able to utilize the existing free spaces inside the layout to create the buffering areas.

In this chapter, we will first define the distance between polygons. Then we will introduce a set of variables that represent the distance between polygons to the position-based optimization model. We then present several floating formulations each offers a different way of distributing the free space. Lastly, we discuss the application of the extension to overlap resolution in a overlapped layout. We show that with the added capability of controlling the distance between polygons, we can partially eliminate overlaps when it is not possible to completely resolve them.

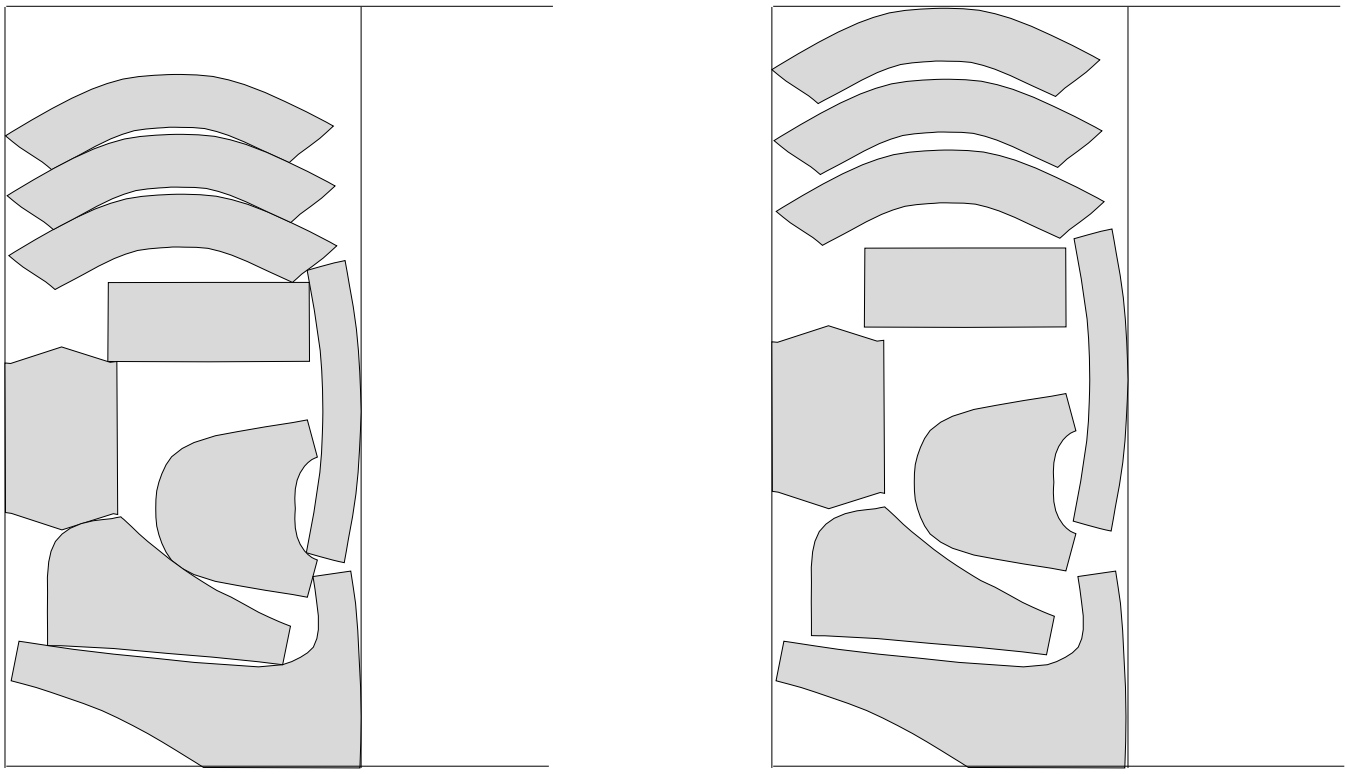


Figure 7.1: An example of floating

7.1 Distance Between Polygons

We begin by giving a formal definition of the distance between two polygons. This notation is a generalization of the "depth of overlap" concept introduced in Chapter 1. We shall see that depth of overlap corresponds to the negative distance between the two polygons. We first define the distance between polygons using the polygons themselves. Then we illustrate how it is reflected in the Minkowski sum of two polygons.

Before we give the definition, we review briefly the incidence relation between two polygons. We say that polygon P *overlaps* polygon Q if there exist some points of P that are contained in the interior of Q (reciprocally, there must also exist some points of Q that are in the interior of P). Otherwise, P and Q are *non-overlapping*. When P and Q are non-overlapping, they can still be *in contact*, that is P and Q are incident on their boundaries.

Again, we use $P(c_P)$ to denote a polygon P whose center is placed at c_P.

Definition 7.1 Given polygons $P(c_P)$ and $Q(c_Q)$, let v be the minimum length vector such that $Q(c_Q + v)$ is in contact with P. The distance between $P(c_P)$ and $Q(c_Q)$, denoted by $D(P(c_P), Q(c_Q))$, is defined as follows:
If $P(c_P)$ and $Q(c_Q)$ do not overlap, then

$$D(P(c_P), Q(c_Q)) = \|v\|$$

If $P(c_P)$ and $Q(c_Q)$ overlap, then

$$D(P(c_P), Q(c_Q)) = -\|v\|$$

where $\|v\|$ is the length of v.

From the definition, we immediately have the following facts.

Fact 7.1 $D(P(c_P), Q(c_Q)) = D(Q(c_Q), P(c_P))$

Fact 7.2 $D(P(c_P), Q(c_Q)) = 0$ *if and only if* $P(c_P)$ *and* $Q(c_Q)$ *are in contact.*

Next, we try to obtain characterization of $D(P(c_P), Q(c_Q))$ from the Minkowski sum $P \oplus (-Q)$. For notational simplicity, let us denote $P \oplus (-Q)$ by M. Let $\eta(c_Q - c_P, M)$ be the minimal distance between $c_Q - c_P$ and δM – the boundary points of M. That is

$$\eta(c_Q - c_P, M) = \ min\ \{\|(c_Q - c_P) - r\| \mid r \in \delta M\}$$

Define $\eta(P(c_P), Q(c_Q))$ as

$$\eta(P(c_P), Q(c_Q)) = \eta(c_Q - c_P, M))$$

if $c_Q - c_P$ is not contained in M. Otherwise

$$\eta(P(c_P), Q(c_Q)) = -\eta(c_Q - c_P, M)$$

Claim 7.3 $D(P(c_P), Q(c_Q)) = \eta(P(c_P), Q(c_Q))$

Proof: Without loss of generality, we only prove for the case where $P(c_P)$ and $Q(c_Q)$ do not overlap. The proof for the case where they overlap is similar, only the signs of $\eta(P(c_P), Q(c_Q))$ and $D(P(c_P), Q(c_Q))$ are changed.

Let b be the boundary point of M that is closest to $c_Q - c_P$. Let w be the vector connecting $c_Q - c_P$ to b. Translating $c_Q - c_P$ by w brings $c_Q - c_P$ to b. By Corollary 3.11, that means that $Q(c_Q + w)$ is in contact with $P(c_P)$. Since, by definition, $D(P(c_P), Q(c_Q))$ is the minimum distance needed to bring $Q(c_Q)$ into contact with $P(c_P)$, we must have

$$D(P(c_P), Q(c_Q)) \leq \|w\| = \eta(P(c_P), Q(c_Q))$$

.

On the other hand, suppose $Q(c_Q)$ is translated by a distance of $D(P(c_P), Q(c_Q))$ to a position $Q(c'_Q)$ to be contact with $P(c_P)$. Again, by Corollary 3.11 , c'_Q must be on the boundary of M. Thus, the distance between c_Q and c'_Q, which is $D(P(c_P), Q(c_Q))$, must be less than or equal to $\eta(P(c_P), Q(c_Q))$ by definition. Hence, we must have

$$D(P(c_P), Q(c_Q)) = \eta(P(c_P), Q(c_Q))$$

□

In the rest of this chapter, we use $D(P(c_P), Q(c_Q)$ and $\eta(P(c_P), Q(c_Q))$ interchangeably to denote the distance between $P(c_P)$ and $Q(c_Q)$.

7.2 Controlling the Distance Between Polygons

Let p and q denote the new positions of P and Q. In Chapter 4, we showed that by finding a large convex subset $\mathcal{C}$ in $\overline{P \oplus (-Q)}$ and constraining $q - p$ to lie in $\mathcal{C}$, we can guarantee that there is no overlap between $P(p)$ and $Q(q)$, or equivalently $D(P(p), Q(q)) \geq 0$. In this

section, we show that by transforming $\mathcal{C}$ into other convex sets and building similar linear constraints on the transformed sets, we are able to force the distance $D(P(p), Q(q))$ to be at least d, where d can be either a constant or a variable.

Let $\mathcal{C}$ be the convex subset identified by the locality heuristic of Chapter 4. Let A_iA_{i+1}, $i = 1, \ldots, k$ be the list of edges of $P \oplus (-Q)$ output by the locality heuristic. By our convention, A_iA_{i+1} is ordered counterclockwise. That is, when walking from A_i to A_{i+1}, the interior of $P \oplus (-Q)$ is on the left-hand side of A_iA_{i+1} and the outward normal of A_iA_{i+1} points to the right-hand side of A_iA_{i+1}.

Associate each edge A_iA_{i+1} with a line $L(A_iA_{i+1})$ that contains A_iA_{i+1}. Let half-plane $H(A_iA_{i+1})$ be delimited by $L(A_iA_{i+1})$ and contain the outward normal of A_iA_{i+1}. Then $\mathcal{C}$ is the intersection of $H(A_iA_{i+1})$ (for $i = 1, \ldots, I$) and is bounded by $L(A_iA_{i+1})$.

Let $L(A_iA_{i+1}, d)$ be obtained by translating the line $L(A_iA_{i+1})$ by a distance d along the outward normal of A_iA_{i+1} and $H(A_iA_{i+1}, d)$ be the corresponding half-plane obtained by translating $H(A_iA_{i+1})$ the same amount. It is clear that $L(A_iA_{i+1}, d)$ is to the right of A_iA_{i+1} for $d > 0$ and to the left of A_iA_{i+1} for $d < 0$.

We now construct a new convex set $\mathcal{C}(d)$ by taking the intersection of $H(A_iA_{i+1}, d)$ for $i = 1, \ldots, I$. It follows immediately that $\mathcal{C}(d)$ is delimited by the lines $L(A_iA_{i+1}, d)$, $i = 1, \ldots, I$.

Clearly, if $d > 0$, then $\mathcal{C}(d)$ is completely contained in $\mathcal{C}$ (see Figure 7.2). In this case, the distance of a point in $\mathcal{C}(d)$ to the boundary of $\mathcal{C}$ is at least d. Since the points of $P \oplus (-Q)$ are either on or outside $\mathcal{C}$, which means the distance of a point in $\mathcal{C}(d)$ to $P \oplus (-Q)$ is at least d because each point is at least d distant from each line containing an edge of the convex polygon. Hence we have,

Lemma 7.4 *Let $d > 0$. If $q - p \in \mathcal{C}(d)$, then $D(P(p), Q(q)) \geq d$.*

7.3 Linear Constraints for Floating

Let $L(A_iA_{i+1}, d)$ be a bounding line of $\mathcal{C}(d)$ with associated half-plane $H(A_iA_{i+1}, d)$. We now consider the linear constraint for $q - p \in H(A_iA_{i+1}, d)$. Again, let us denote $q - p$ by R. From Appendix A, we know that the cross product $A_iR \times A_iA_{i+1}$ equals twice the signed area of the triangle $\triangle RA_{i+1}A_i$. The signed area is positive when R, A_{i+1} and A_i are ordered counterclockwise, or, equivalently, when $R \in H(A_{i+1}A_i)$. On the other hand, the area of $\triangle RA_{i+1}A_i$ equals $\frac{1}{2}h\|A_iA_{i+1}\|$, where h is the signed distance from R to $L(A_iA_{i+1})$.

We have

$$h = 2\frac{A_iR \times A_iA_{i+1}}{\|A_iA_{i+1}\|}.$$

Hence, $R \in H(A_iA_{i+1}, d)$ can be expressed as

$$2\frac{A_iR \times A_iA_{i+1}}{\|A_iA_{i+1}\|} \geq d \tag{7.1}$$

Since $A_iR \times A_iA_{i+1}$ is a linear constraint on R, the inequality 7.1 is a linear constraint in the variables p, q and d. It is important that d is a *variable* in the linear constraint. It enables us to have flexible control over the distance between polygons.

7.4 Maximize the Minimum Distance Between Polygons

With the linear constraint 7.1 built in the previous section, we immediately have an algorithm for maximizing the minimum distance between the polygons. We obtain this algorithm by modifying the position-based optimization model (see Chapter 4).

In the modified model, we replace the original linear constraints with the linear constraints of the form 7.1. The objective function is simply to maximize d. If we assume that there is no overlap in the layout, then the current positions of the polygons and the value $d = 0$ always form a feasible solution of the model. If the model has an optimal solution $d^* > 0$, then by Lemma7.4, the distance between a pair of neighboring polygons is at least d^* when the polygons are placed at their new positions.

7.5 Separating Polygons by a Specific Distance

Here, by "separating polygons by a distance d" we mean to find a set of new position of the polygons in a layout in which the distance between every pair of polygons is at least d. If we are only interested in detecting whether the polygons in a layout can be separated by a specific distance $\bar{d}$, we can add a constraint $d = \bar{d}$ to the model in the previous section and use a constant objective function. With the constant objective function, the linear program solver will return a set of feasible solutions immediately after it finds one. That is, the linear program solver will stop once it reaches the boundary or the interior the feasible region without marching to the optimal point. Hence, it is faster to solve.

Moreover, we can combine the above formulation with the one in Section5.2 to achieve the result of separating the polygons by a distance of $\bar{d}$ while minimizing the total translations of the polygons. To do so, we replace the constant objective function in the previous

paragraph with the following objective function.

$$minimize \quad \sum_{i=1}^{n} |q_i|$$

We use the same technique as in Section 5.2 to eliminate the absolute values.

7.6 Maximizing the Overall Distance between Polygons

If the model in Section7.4 does not have a non-zero optimal solution, it is a good indication that there is a critical set in the layout. We say a list of polygons $P_1, P_2, \ldots, P_l$ forms a *critical set* in a layout if P_i is in contact with P_{i-1} and P_{i+1} $(2 \leq il - 1)$ and it is impossible to make the distance between P_i and P_{i+1} $(1 \leq i \leq l - 1)$ greater than zero without increasing the length of the layout.

For layouts with critical sets, although the polygons that are included in some critical set can not be moved, there can still be some free areas that can be "distributed" among the polygon not in the critical sets. Hence, we need a formulation that achieves partial results. To do so, we revise the model in Section 7.4 by assigning a separate variable d_j for the jth pair of neighboring polygons for $j = 1, \ldots, J$, where J is the number of neighboring polygons found by the sweepline algorithm. We call these variables *distance* variables and they are subject to the bounds $d_j \geq 0$ $(j = 1, \ldots, J)$. When building the constraints for the j'th pair, we replace the d in inequality 7.1 by d_j. The objective function is to maximize the overall distance between neighboring pairs of polygons:

$$maximize \sum_{j=1}^{J} d_j$$

With the above formulation, the distance variables corresponding to the pairs of neighboring polygons that are in some critical set will be zero in the optimal solution. However, other distance variables can now have non-zero values and hence those polygons will be separated by various distances.

7.7 "Uniform" Distribution of Free Areas

We see that the formulations in Section 7.4 and Section 7.6 each have their own advantages and drawbacks. For the formulation in Section 7.4, free space in the layout are distributed among polygons uniformly in the sense that every pair of adjacent polygon is

separate by at least a distance d^* for an optimal solution $d^* > 0$. But, it may fail to increase the distance among polygons if there exist critical sets in the layout. The formulation in Section 7.6, on the other hand, is usually able to find the optimal solution which has non-zero value for some of the distance variables. However, since the objective function is to maximize the sum of all the distance variables, it may increase the distance between some pairs of polygons at the expense of decreasing the distance between some other pairs of polygons or keep those distances very small. This behavior may not be desirable for certain applications.

In this section, we find a compromise between the two formulations. We adopt the linear constraints from Section 7.6 and achieve our goal by modifying the objective function.

Let us create a new variable m to represent the mean of all the distance variables.

$$m = \frac{1}{J} \sum_{j=1}^{J} d_j$$

Our first goal is the same as the one in the formulation of Section 7.6 which is to maximize the overall distances between the neighboring polygons. Hence, we would include

$$maximize \quad Jm$$

or, equivalently,

$$minimize \quad -Jm$$

in our new objective function.

Our second goal is to distribute the free spaces as uniformly among the polygons as possible. That is, we want to minimize the deviation between a distance variable and m. We express this goal using the following objective

$$minimize \quad |d_1 - m| + |d_2 - m| + \ldots + |d_c - m|$$

Combining the two goals together, we obtain our new objective function[1]:

$$minimize \quad -Jm + \sum_{j=1}^{J} |d_j - m| \tag{7.2}$$

[1]Note that our ability to express and combine our objectives is limited by the linear program model, which only allows a single linear objective function. Hence, we have to express each of our individual goal as a linear function and combine them using a linear combination. The single objective function we have devised does not necessarily accomplish the multiple objectives exactly the ways we intended it to. However, we can influence the behavior of the model by assigning proper weights in the linear combination.

We now use the same technique as in Section 5.2 to eliminate the absolute value in the objective function. We first introduce $2J$ new variables l_i^+ and l_i^- for $1 \leq j \leq J$. Next, we replace the term $|d_j - m|$ in the objective function by $l_i^+ + l_i^-$ and add the following constraints

$$l_i^+ - l_i^- = d_i - m$$

and bounds:

$$\begin{aligned} l_i^+ &\geq 0 \\ l_i^- &\geq 0 \end{aligned}$$

into the linear program.

Remark 7.1 We can have an alternative technique to eliminate the absolute value in the objective function 7.2. In this technique, we introduce J new variables l_j for $1 \leq j \leq J$ and replace $|d_j - m|$ in the objective function by l_j. Then, we add the following constraints:

$$\begin{aligned} d_j - m &\leq l_j \\ d_j - m &\geq -l_j \end{aligned}$$

and bounds

$$l_j \geq 0$$

to the linear program.

The difference with the previous technique is that we introduce J fewer variables but J more constraints. Empirical studies show that the simplex method for solving a linear program runs approximately in time linear in $max(m, n)$, where m is the number of variables and n the number of constraints. Since we usually have many more constraints than variables, we do not expect significant difference in the performance of the two techniques.

7.8 Maximize the Minimum Distance Between Polygons: An Alternative Method

The formulations in previous sections try to achieve their goals in a single step by "designing" a suitable objective function. The linear objective function itself offers limited flexibility. Sometime, when the goal cannot be achieved by specifying a single objective

function, we may try to set up linear program with different objective functions as subroutines and use an algorithm to control these subroutines. For example, when there exists critical sets in a layout, the formulation in Section 7.4 fails to increase the distance between any pairs of polygons because d^* is zero.

We can actually solve the problem in two steps. In the first step, we set up the objective function to maximize the sum of d_i's. Then we identify the polygons in the critical sets by checking whether the corresponding distances in the solution are zeros. In the second step, we can just maximize the minimum separation among the non-critical polygons.

7.9 Floating For Overlapped Layouts

Since the linear constraint we formulated in Section 7.3 is an extension to the one in Section 4.1.1, all of the formulations presented in previous sections can be applied to a layout that contains overlaps. In particular, with d set to 0, the formulation in Section 7.5 exactly the same as the original formulation for separation presented in Section 5.2.

Moreover, with the distance variables, we have more flexible control over elimination of overlaps. In contrast to the model in Section 5.2 which fails to make any improvement when it cannot eliminate all the overlaps in a layout, we can now eliminate the overlaps partially, that is eliminate the overlaps for some of the overlapped pairs of polygons but not all of them. We can also make improvements by reducing the negative distance between two overlapped polygons.

However, to make the formulations work well on layouts that contain overlaps, we need to make the following modification to the formulations. The modification is to replace the bounds

$$d_j \geq 0 \quad (j = 1, \ldots, J)$$

in the previous linear program models with

$$d_j \geq -\bar{d}_j \quad (j = 1, \ldots, J)$$

for some proper constants $\bar{d}_j > 0$ $(j = 1, \ldots, J)$ to be specified next.

There are two conflicting requirements for $\bar{d}_j$ $(j = 1, \ldots, J)$. First, $\bar{d}_j$ should be small in order to prevent d_j from having large negative value, which means deep overlap between the two polygons in the jth pair. However, if $\bar{d}_j$ is too small, we might not be able to find any feasible solutions. Therefore, we want $\bar{d}_j$ to be big. In particular, for partial resolution of overlaps, it is important for the $\bar{d}_j$ to have big enough values such that the current

positions of the polygons (plus any value of d_j that satisfies $d_j \geq \bar{d}_j$ for $1 \leq j \leq J$) can be a feasible solution of the linear program model. The reason is that we want to make possible any infinitesimal motion of polygons that could result in an improvement in the objective function. If the current positions of the polygons cannot form a feasible solution, those infinitesimal movements may not be possible.

We now present a method for choosing the bounds $\bar{d}_j$ ($j = 1, \ldots, J$) that satisfy the aforementioned requirements. Let us assume that the jth pair of polygons $P(c_P)$ and $Q(c_Q)$ are overlapping each other. Let $\mathcal{C}$ be the convex subset of $\overline{P \oplus (-Q)}$ found by the locality heuristic. Since $P(c_P)$ and $Q(c_Q)$ are overlapping, the point $c_Q - c_P$ is not contained in $\mathcal{C}$. Intuitively, we want to "expand" $\mathcal{C}$, that is, move the bounding lines of $\mathcal{C}$ towards the exterior of $\mathcal{C}$ in equal speed, until one of the lines hits $c_Q - c_P$. Let $\bar{d}_j$ be the distance traveled by the bounding lines (see Figure 7.3). More precisely, we want to find the convex set $\mathcal{C}(-h)$ for $h > 0$ such that $c_Q - c_P$ is on the boundary of $\mathcal{C}(-h)$ and assign h to $\bar{d}_j$. Such an h can be found by calculating the signed distance of $c_Q - c_P$ to the bounding lines of $\mathcal{C}$. The distance of $c_Q - c_P$ to a bounding $L(A_iA_{i+1})$ of $\mathcal{C}$ is positive if $c_Q - c_P$ is contained in the half-plane $H(A_iA_{i+1})$, otherwise it is negative. Then, $-h$ is chosen as the smallest (negative) value of the signed distances. Clearly, with $-h$ and $\bar{d}_j$ so chosen, c_P and c_Q satisfy the constraints built for $P(c_P)$ and $Q(c_Q)$. Therefore, if all $\bar{d}_j$ ($1 \leq j \leq J$) are chose this way, the current positions of the polygons form a feasible solution of the linear program model.

We next argue that the $\bar{d}_j$ ($j = 1, \ldots, J$) chosen as described above is in some sense the set of smallest possible values to guarantee that the current positions form a feasible solution. The reason is simply that if $\bar{d}_j$ is smaller than the h of the previous paragraph, then we need to move $c_Q - c_P$ by a non-zero distance to satisfy the constraints built for $P(c_P)$ and $Q(c_Q)$ which corresponds to $\mathcal{C}(\bar{d}_j)$. But that motion may not be possible. Hence, the current positions cannot form a feasible solution. In this event, the infeasibility imposed by a small d_j prevents possible improvements in the other part of the layout.

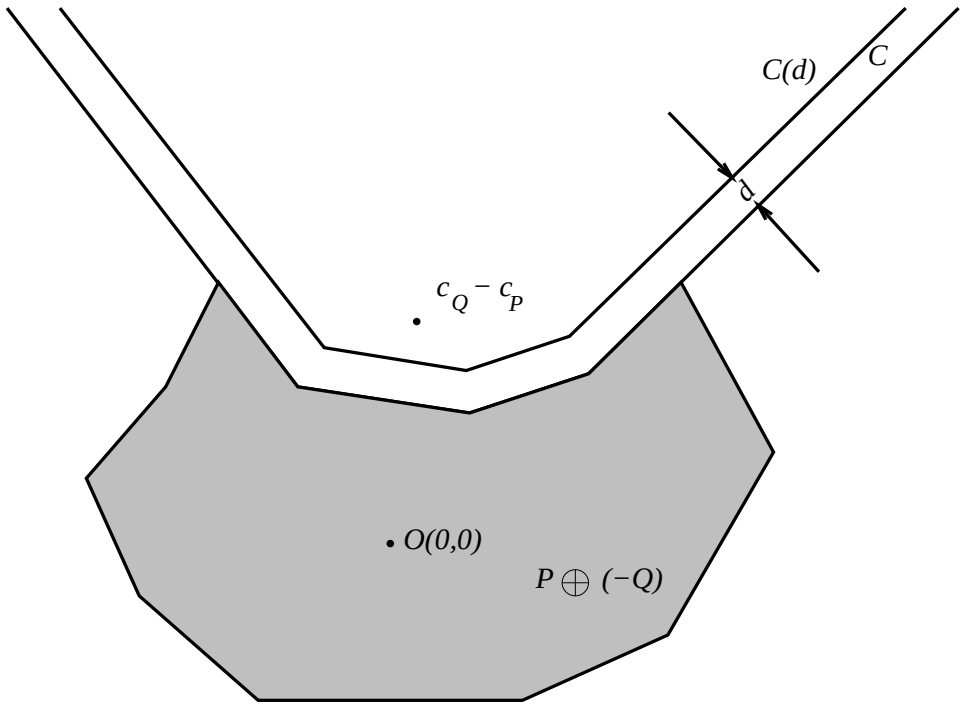


Figure 7.2: $\mathcal{C}(d)$ — the convex set obtained by translating the boundary lines of $\mathcal{C}$ by a distance of d

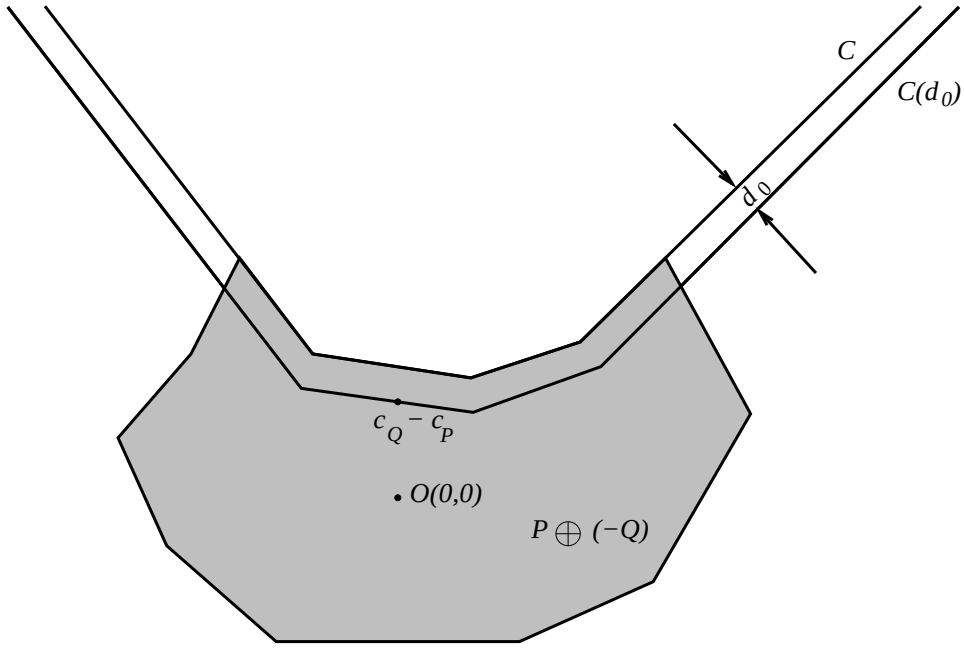


Figure 7.3: The translated convex set for slightly overlapped polygons

Chapter 8

Mixed Integer Programming Model for Compaction and Two-Dimensional Packing

In this chapter, we extend the linear programming formulation used in our position-based optimization model to a mixed integer programing (MIP) formulation. The MIP formulation offers a reduction of the two-dimensional strip packing problem (see Chapter 1) to the MIP problem.[1] Hence, the optimal solution of a two-dimensional strip packing problem can be found by solving a mixed integer program. The reduction also serves as a direct proof that optimal 2D strip packing problem is in NP.

The MIP formulation is then further extended to solve the multiple containment problem: the problem of finding a non-overlapping placement of a set of non-convex polygons in a non-convex container. In automated marker making applications, this problem corresponds to placing several trim pieces in a gap. Using the MINTO optimization package, we have sometimes been able to place up to nine polygons in a gap in a few minutes.

8.1 Limitation of the Locality Heuristic

In our position-based compaction algorithm, the locality heuristic is used to restrict the relative motion of polygon Q with respect to polygon P to a convex subset of $\overline{P \oplus (-Q)}$. On the one hand, such a restriction is essential in reducing the motion planning problem

[1] Flips and discrete rotations can be added to the MIP model, but we have not attempted to implement this.

embodied in compaction to a more manageable linear programming problem. On the other hand, it may exclude other solutions to a compaction problem that may result in a shorter length. A simple example is depicted in Figure 8.1. Figure 8.1 (a) shows the initial configuration of a layout to be compacted. Figure 8.1(b) shows the result of applying our position-based compaction algorithm.

Obviously, for the layout in Figure 8.1(a), the best result that can be achieved by a compaction algorithm is a layout of length l in which polygon Q and polygon P are stacked vertically. The actual result achieve by our position-based compaction algorithm has length $2l - 1$.

In this example, our position-based compaction algorithm failed to find the optimal compaction result of length l because the convex subset selected by the locality heuristic prevents P and Q from moving to the relative position that yields the optimal solution. The starting edge selected by the locality heuristic is the edge b of $P \oplus (-Q)$. The convex subset $\mathcal{C}$ of $\overline{P \oplus (-Q)}$ built from b is actually the half-plane delimited by b and on the right-hand side of b. The linear constraint built from $\mathcal{C}$ constrains Q to stay on the right of P. Thus, the length $2l - 1$ of the layout shown in Figure 8.1(b) is the best length achievable given the constraint.

Clearly, if the edge a of $P \oplus (-Q)$ (see Figure 8.1(c)) is selected as the starting edge for building a convex subset $\mathcal{C}_1$ of $\overline{P \oplus (-Q)}$, then $\mathcal{C}_1$ is the half-plane delimited by and above a. The linear constraint built from $\mathcal{C}_1$ states that Q must lie on top of P. Hence, using $\mathcal{C}_1$ instead of $\mathcal{C}$ to build the non-overlapping constraints would enable our position-based compaction algorithm find the optimal compaction of length l.

An immediate idea to improve our position-based compaction algorithm is therefore to try several convex subsets of $\overline{P \oplus (-Q)}$ for each pair of polygons P and Q and choose the ones that collectively yield the best compaction result. However, the following theorem shows that making such a choice is NP-hard.

Theorem 8.1 *Given a layout consisting of N polygons P_1, P_2, ..., P_N. Assume that for each pair of polygons P_i and P_j ($1 \leq, i, j \leq N$, $i \neq j$), we can choose one from two different convex subsets of $\overline{P_i \oplus (-P_j)}$ to build non-overlapping constraints for P_i and P_j in our position-based compaction algorithm. Then, finding a combination of the convex subsets that achieves the optimal solution for compaction is NP-hard.*

Proof: We again reduce the now familiar PARTITION problem to our problem. The reduction is shown in Figure 8.2. For an instance $\{a_1, a_2, \ldots, a_N\}$ of PARTITION, we

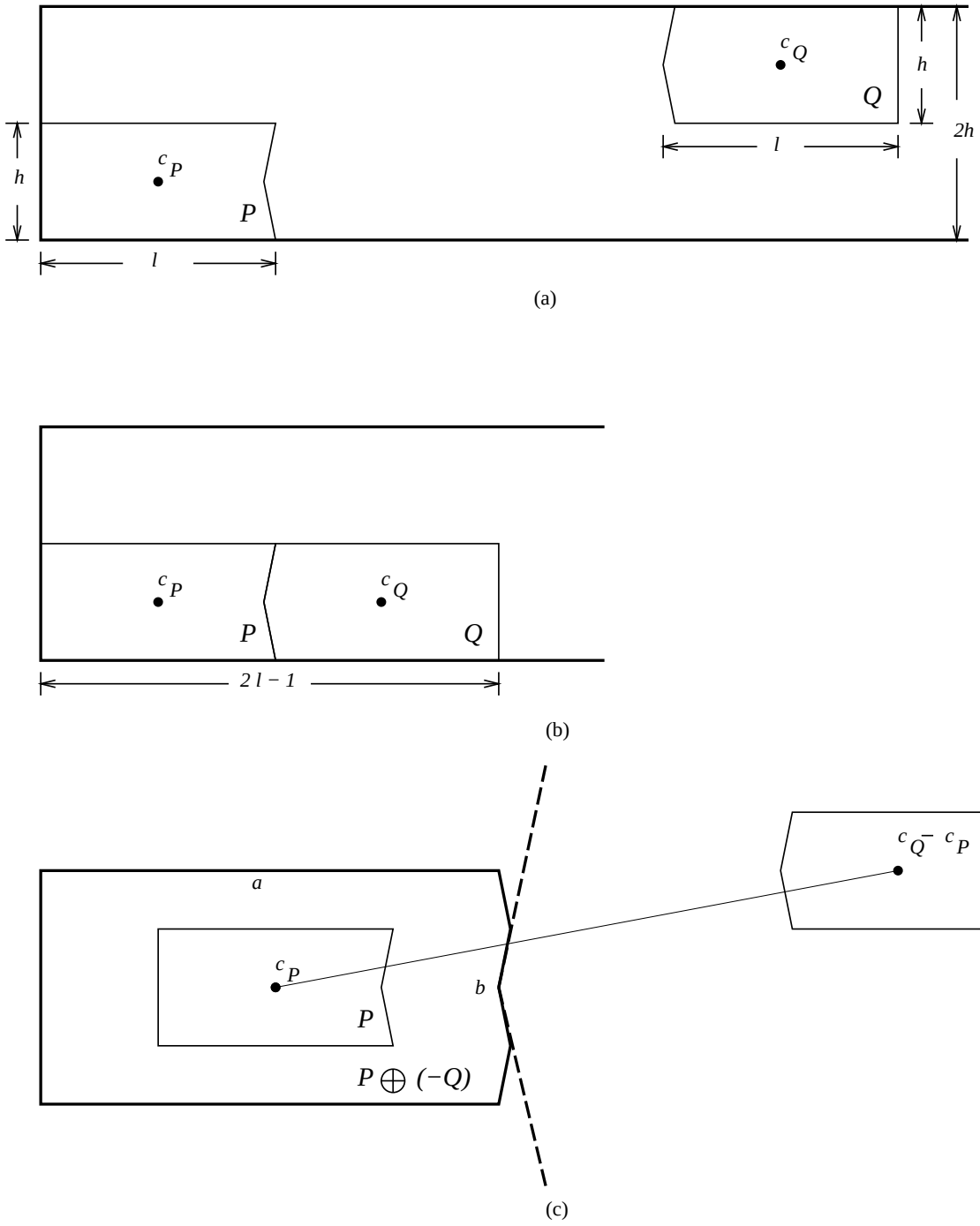


Figure 8.1: (a) The initial layout. (b) The compaction result using the position-based compaction algorithm. (c) Illustration of the locality heuristic: the dotted lines illustrates the boundary of the convex subset identified by the locality heuristic.

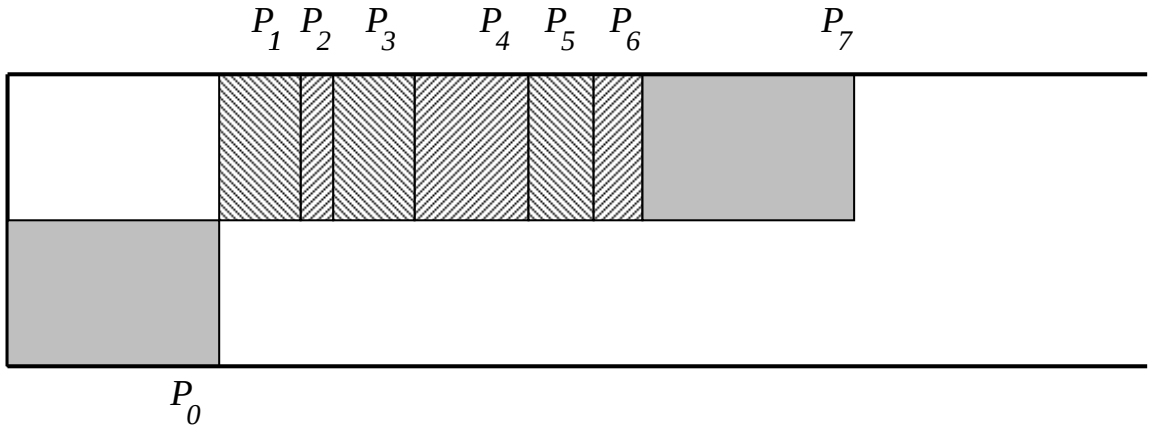


Figure 8.2: Reduction of PARTITION to selecting the right combination of convex subsets.

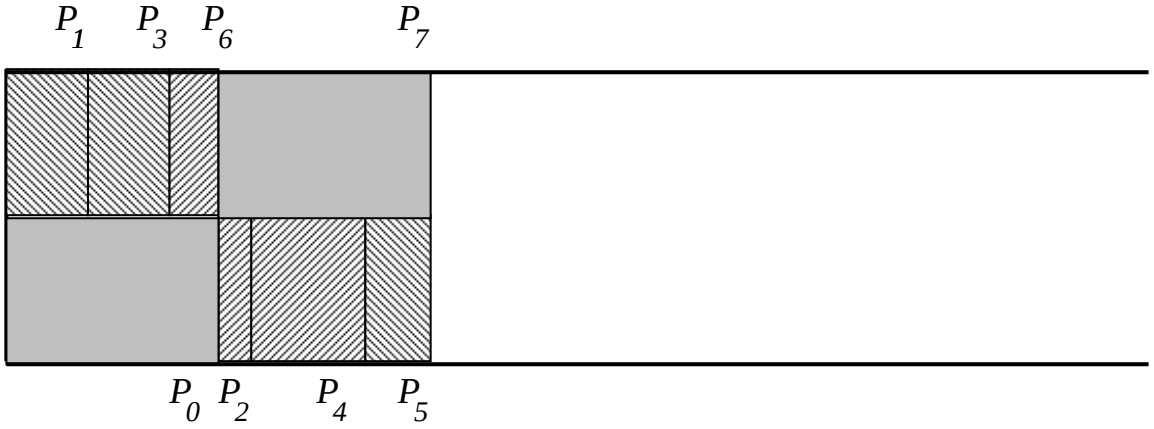


Figure 8.3: Successful selection of the right combination of convex subsets solves PARTITION.

build rectangles P_i of height 1 and length a_i, $(i = 1, \ldots, N)$. We then add two additional rectangles P_0 and P_{N+1} each of height 1 and length $B = \frac{1}{2}\sum_{i=1}^{N} a_i$. Initially, these polygons are placed in the configuration shown in Figure 8.2.

For rectangles P_i and P_j $(i \neq j)$, the Minkowski sum $P_i \oplus (-P_j)$ is also a rectangle. Let t, b, l and r represent the top, bottom, left and right edges of $P_i \oplus (-P_j)$ respectively. Let convex sets H_t, H_b, H_l and H_r be the half-planes in $\overline{P_i \oplus (-P_j)}$ that are delimited by edge t, b, l and r respectively. The non-overlapping constraint built using H_t constrains P_j to stay above P_i. The other three cases can be deduced similarly.

The choices of convex sets for pairs of polygons are as follows. First, for building non-overlapping constraints of P_i $(1 \leq i \leq n)$ with P_0, we offer two choices of convex subsets:

the one that constrains P_i to stay on top of P_0 and the one that constrains P_i to stay to the right of P_0. Second, for building non-overlapping constraints between P_i $(1 \leq i \leq n)$ and P_{n+1}, we offer the convex subset that constrains P_i to stay to the left P_{n+1} and the one that constrains P_i to stay below P_{n+1}. Third, for building non-overlapping constraints between P_i and P_j $(1 \leq i, j \leq n,\ i < j)$, we offer the convex subset that constrains P_i to stay to the left P_j and the one that constrains P_i to stay below P_j.

Figure 8.3 shows the minimal length compaction of the layout in Figure 8.2. It is clear that PARTITION has a solution if and only if the layout in Figure 8.2 has a length $2B$ compaction. □

In the next section, we will use a set of Boolean variables (each has value 0 or 1) to represent each of the convex subsets to be chosen. The problem of choosing the combination of convex subsets that yields the minimal length compaction is then reduced to a MIP problem. Theorem 8.1 illustrates that using the full power of MIP is justified since the problem is at least as hard as MIP (that is, NP-hard) in the worst case. More importantly, reducing the problem to MIP enables us to take the advantage of the powerful techniques developed recently for solving mixed integer programs, especially the techniques deployed by the MINTO package [Nem93] which we are currently using.

Remark 8.1 Note that in Figure 8.1(a), if the upper right vertex of P and the lower left vertex of Q are incident, then the point $c_Q - c_P$ is incident on the vertex at the upper right corner of $P \oplus (-Q)$. In this case, we are forced to make an arbitrary choice between using edge a or edge b to build a non-overlapping constraint. We can use the same idea as in the proof of Theorem 8.1 to show that making a consistent choice that leads to minimal length compaction for layouts that have vertex-vertex incidences is NP-hard.

8.2 MIP Formulation for Optimal Two-Dimensional Packing/Compaction

From the previous section, we see that in the optimal solution of compaction, the relative position of Q with respect to P does not necessarily lie in the convex subset of $\overline{P \oplus (-Q)}$ identified by the locality heuristic. In that case, our position-based compaction algorithm fails to find the optimal solution since it restricts the relative motion of Q with respect to P to stay in the convex subset found by the locality heuristic.

Thus, in order to find the optimal solution of compaction, we need to consider the whole free space $\overline{P \oplus (-Q)}$ for each pair of polygons P and Q. To this end, we first give the following definitions.

Definition 8.1 *Let S be a set and let $c_1, c_2, \ldots, c_k$ be subsets of S. We call $c_1, c_2, \ldots, c_k$ a* **covering** *of S if*

$$S = \bigcup_{i=1}^{k} c_i$$

We call $c_1, c_2, \ldots, c_k$ a **convex covering** *of S if $c_1, c_2, \ldots, c_k$ is a covering of S and c_i is a convex set for $1 \leq i \leq k$. We call $c_1, c_2, \ldots, c_k$ a* **convex decomposition** *of S if $c_1, c_2, \ldots, c_k$ is a convex covering of S and $c_i \cap c_j = \emptyset$ for $1 \leq i \leq k$.*

We now extend our position-based model to include multiple convex subsets that cover the free space for the motion planning of a pair of polygons. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ be a convex covering of $\overline{P \oplus (-Q)}$ for polygons P and Q. For each subset $\mathcal{C}_i$, we can induce a set of linear constraints on $q - p$, the new relative position of Q with respect to P, based on the boundaries of $\mathcal{C}_i$ (see Section 4.1.2). Each of the linear constraint has the following form.

$$\alpha q_x + \beta q_y + \gamma p_x + \delta p_y \geq \eta \tag{8.1}$$

where $\alpha, \beta, \gamma, \delta$ and η are constants.

Let us assign a Boolean variable B_i to each of the convex subsets $\mathcal{C}_i$ $(1 \leq i \leq k)$. These Boolean variables satisfy

$$\sum_{i=1}^{k} B_i = 1.$$

That is, at any moment only one of the Boolean variables can be true. If B_i is 1, we say that $\mathcal{C}_i$ is *selected*. Otherwise, $\mathcal{C}_i$ is *not selected*.

Assume 8.1 is a linear constraint built from convex subset $\mathcal{C}_i$. We expand the linear constraint to include the Boolean variable B_i.

$$\alpha q_x + \beta q_y + \gamma p_x + \delta p_y \geq (B_i - 1)\lambda + \eta \tag{8.2}$$

where λ is a sufficiently large constant such that when B_i is 0, constraint 8.2 is satisfied automatically.

Thus, among all the constraints built from the convex subsets only the constraints built from the selected convex subset are effective in constraining $q - p$. All the other constraints become redundant. By selecting a proper convex subset, we are able to place $q - p$ anywhere in the free space $\overline{P \oplus (-Q)}$.

We now perform the above steps for every pair of polygons in a layout. The result is an extension of the position-based compaction algorithm that is capable of finding the global optimal of compaction. The following is the modified algorithm.

A position-based compaction algorithm using MIP formulation.

Input "force" f_i for polygon i $(i = 1, \ldots, N)$; /* f_i's are constants */

Assign a positional variable p_i for polygon i $(i = 1, \ldots, N)$;

do

L := the list of all pairs of polygons;

S := $\emptyset$; /* S is the set of linear constraints generated so far */

foreach pair of polygons (P, Q) in L **do**

Compute $\overline{P \oplus (-Q)}$;

Find a convex covering $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ of $\overline{P \oplus (-Q)}$;

Assign a Boolean variable B_i for $\mathcal{C}_i$ $(1 \leq i \leq k)$;

S_0 := the MIP constraints generated from $\mathcal{C}_i$ $(1 \leq i \leq k)$ according to (8.2);

S := $S \cup S_0$;

Add the constraint $\sum_{i=1}^{k} B_i = 1$;

end

foreach polygon P **do**

S_1 := the non-overlapping constraints between P the boundaries of the container;

S := $S \cup S_1$;

end

Set up a mixed integer program

maximize $\sum_{i=1}^{N} f_i \cdot p_i$

subject to S;

Get the set of new positions p_i $(i = 1, \ldots, N)$ by solving the mixed integer program;

Place polygon i at p_i $(i = 1, \ldots, N)$;

until the objective function $\sum_{i=1}^{N} f_i \cdot p_i$ stops increasing.

It follows from the algorithm that the following theorem holds.

Theorem 8.2 *A feasible solution to the MIP model built in the above algorithm gives a non-overlapping placement of the polygons in the container. Conversely, from a non-overlapping placement of the polygons, we can obtain a feasible solution to the MIP model.*

The following theorem is the main result of this section.

Theorem 8.3 *The position-based compaction algorithm using MIP formulation solves the two-dimensional strip packing problem*[2] *for non-convex polygons.*

[2]Assume the polygons can not rotate or flip.

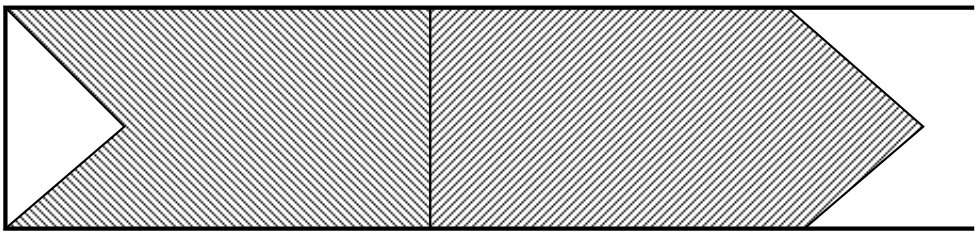


Figure 8.4: Length cannot be shortened without exchanging the positions of the polygons

Proof: As before, we view the right boundary of the container as the front-end of a "piston" piece and assign a positional variable p_{N+1} to it. The constraints of the polygons with the container ensure that every polygon lies to the left of the piston piece. The objective function of the MIP model becomes

$$minimize \quad p_{N+1}.$$

Let l^* be the length of the optimal solution of a two-dimensional strip packing problem. Since the optimal solution of the two-dimensional strip packing is also a non-overlapping placement of the polygons, from the previous theorem it has a corresponding feasible solution of the MIP model. Therefore, p^*_{N+1}, the optimal solution of the MIP model for a layout, is of equal or lesser length than l^*. But p^*_{N+1} cannot be smaller since it is already optimal. Therefore p^*_{N+1} equals l^*. □

We note here that in the solution of the MIP model two polygons can exchange positions, attaining positions which might not be possible given only continuous motion of the two polygons. Therefore the optimal solution of the MIP model is foremost the optimal solution of the two-dimensional strip packing problem. Whether it is also an optimal solution to the compaction problem depends on whether there is continuous motion that translates the polygons from their initial positions to the positions given by the optimal solution of the MIP. If such a continuous motion does not exist, then the optimal solution to the compaction may have longer length than the optimal solution to the strip packing. As we shall see in the next chapter, finding the optimal solution to compaction is harder than finding the optimal solution to strip packing.

For example, for the layout shown in Figure 8.1(a), the optimal solution for strip packing is also a solution for compaction because we can push polygon Q all the way to the left. However, for the layout shown in Figure 8.4, exchanging the position of the two polygons,

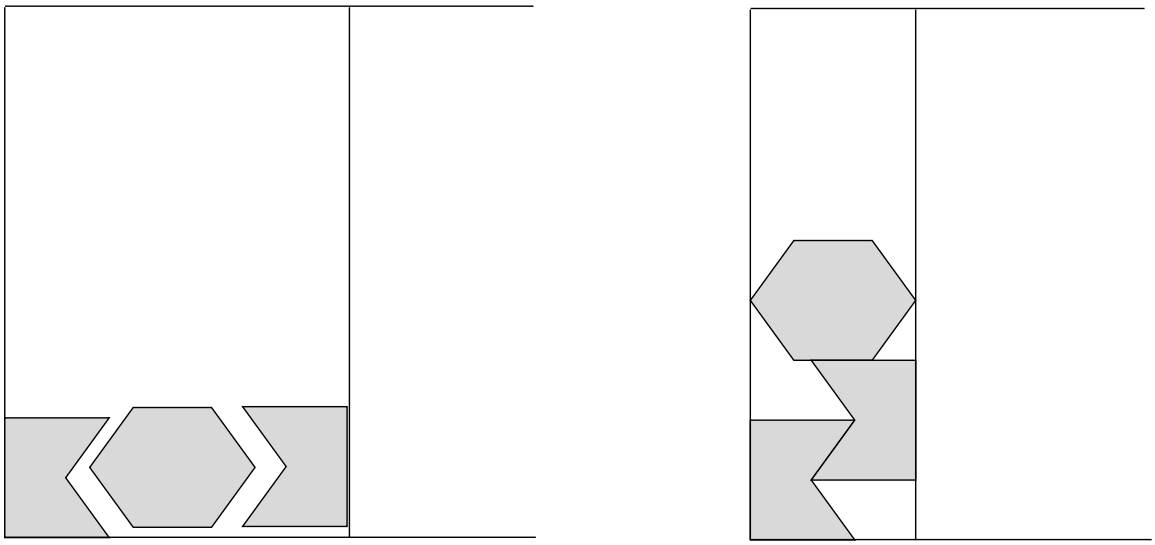


Figure 8.5: Example of finding optimal compaction using MIP formulation

which is the only way to improve the length of the layout, can not be achieve by continuous coordinated motion of the two polygons.

Figure 8.5 shows an example of finding the optimal compaction using the position-based compaction with MIP formulation.

8.2.1 Algorithms for Finding Convex Covering

We now specify how to construct the convex subsets $\mathcal{C}_i$ $(1 \leq i \leq k)$ which cover the free space $P \oplus (-Q)$. The general problem of decomposing of polygon into minimal number of convex polygons has been studied in computational geometry. There exists a polynomial time algorithm to decompose a simple polygon into minimal the number of convex polygons [Kei85]. Note that a linear number of convex polygons is sufficient since triangulation is a convex decomposition.

For simplicity, we assume that $P \oplus (-Q)$ does not have holes. To apply the existing algorithms, we make a rectangle that contains $P \oplus (-Q)$. The rectangle is made sufficiently large such that we will not encounter the need to place $q-p$ outside the rectangle during the Compaction/Packing algorithm. We then take out $P \oplus (-Q)$ from the rectangle. The result is a polygon with a hole that approximates $\overline{P \oplus (-Q)}$. We can use a simple transformation to convert the polygon with a hole to a simple polygon and then decompose the converted simple polygon. If $P \oplus (-Q)$ has multiple holes, the process is similar.

If $P \oplus (-Q)$ is starshaped, then we have a simple and more direct algorithm of finding

a convex covering of the whole free space $\overline{P \oplus (-Q)}$. The algorithm proceeds as follows. Let us number the edges of $\overline{P \oplus (-Q)}$ as $e_1, e_2, \ldots, e_L$. We start from edge e_1. We use e_1 as a starting edge in the locality heuristic to find a convex subset. Once a convex subset is found, we output the subset and mark all the edges that appear in subset. Then, we start from the next unmarked edge and use it as the starting edge of the next convex subset. We stop when all the edges have been marked. In the worst case, such an algorithm runs in $O(n^2)$ time as shown in Figure 8.7, where n is the total number of edge in the polygon.

Theorem 8.4 *If $P \oplus (-Q)$ is starshaped, then the above algorithm generates a covering for $\overline{P \oplus (-Q)}$.*

Proof: Let U be an arbitrary point in $\overline{P \oplus (-Q)}$. Let O be a kernel point of $\overline{P \oplus (-Q)}$. Since $P \oplus (-Q)$ is a simple Jordan curve and O is inside $P \oplus (-Q)$ and U is outside $P \oplus (-Q)$, the edge segment UO either intersects the boundary of $P \oplus (-Q)$ at one point or contains an edge of the boundary. Let us only consider the first case; the second case is similar. Let I be the single intersection point of UO with the boundary of $P \oplus (-Q)$ (see Figure 8.6). Then, the edge segment UI is completely contained in the convex subset $\mathcal{C}$ that contains I. Assume otherwise, that is, $\mathcal{C}$ does not totally contain UI. Then there must be an boundary edge of $\mathcal{C}$ that intersects UI at a point I_1. Then I_1 is not visible from the kernel point since it is hidden by I. This contradicts the fact that $P \oplus (-Q)$ is a starshaped polygon. □

The polynomial time decomposition and covering algorithms establish that the reduction of two-dimensional strip packing to MIP via our position based algorithm can indeed be done in polynomial time and the resulting MIP model has size polynomial in the input size of the strip packing problem. Hence, this provides another proof that strip packing of non-convex polygons is in NP. Another proof that strip packing of arbitrary shaped polygons is in NP is given in [MDL91].

8.3 MIP Formulation for Multiple Containment Problem

Given a polygonal container Q and a set of polygons $P_1, P_2, \ldots, P_k$, the multiple containment problem asks for a non-overlapping placement of $P_1, P_2, \ldots, P_k$ inside Q. We again try to find a MIP formulation for this problem.

To solve the multiple containment problem, we must take into consideration two types of non-overlapping constraints. The first type is the non-overlapping constraints between P_i

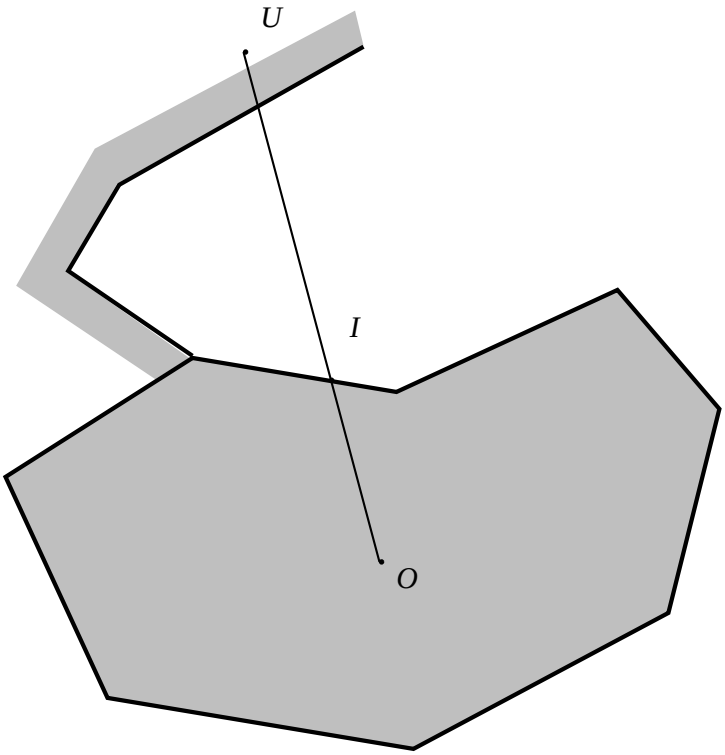


Figure 8.6: Correctness of the convex covering algorithm.

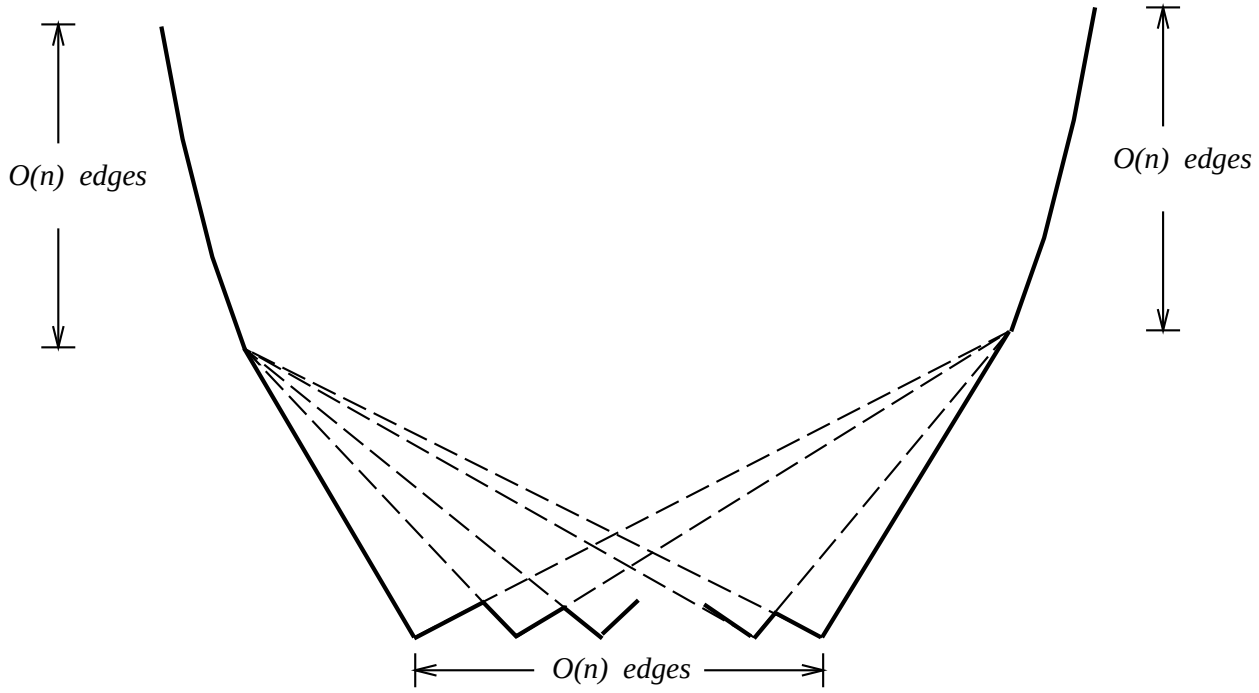


Figure 8.7: Quadratic running time of the convex covering algorithm.

and P_j $(i \neq j)$. The constrains are built from the Minkowski sum $P_i \oplus (-P_j)$. The second type is the non-overlapping constraints between P_i and the polygonal container Q. These constraints are built from the Minkowski differences $Q \ominus (-P_i)$. The MIP formulation for the first type of non-overlapping constraints has been studied in the previous section. In this section, we concentrate on the second type of non-overlapping constraint.

We apply the same basic idea as in the previous section. That is, since $Q \ominus (-P_i)$ is generally non-convex, we decompose $Q \ominus (-P_i)$ into several convex subsets and use a set of Boolean variables to select one of them.

Let $\mathcal{C}^1, \mathcal{C}^2, \ldots, \mathcal{C}^k$ be a convex decomposition (or covering) of $Q \ominus (-P_i)$. We again assign each of these convex subsets a Boolean variable B^i. The Boolean variables are incorporated into the linear constraints built from the convex subsets using the same method as in the linear constraint 8.2.[3] By selecting a proper B^i $(1 \leq i \leq n)$, p_i, the new position of P_i can be placed anywhere in $Q \ominus (-P_i)$.

The position-based multiple containment algorithm is the same as the position-based compaction algorithm using MIP formulation in the previous section. The modification we need to make is to replace the simple linear constraints between P_i and the rectangular container with the mixed integer constraints between P_i and a polygonal container Q. Similar to Theorem 8.2, we have

Theorem 8.5 *A feasible solution to the MIP model built in the position-based multiple containment algorithm gives a non-overlapping placement of the polygons in the polygonal container.*

Since, in multiple containment problem, all we interested in is a feasible solution, we use a dummy (constant) objective function to replace the "energy" function in our normal formulation. With the constant objective function, the mixed integer program solver returns immediately after it finds the first feasible solution, without evaluating further feasible solutions. We observed that the use of a constant objective function reduced the running time significantly.

We have applied the multiple containment algorithm to trim placement. Figure 8.8 shows an example of placing four polygons in a gap. In [DLM94], the authors compared the MIP based algorithm for multiple containment problem with the more purely geometric algorithms.

[3] However, since Q is a static object here, we have $q_x = 0$ and $q_y = 0$.

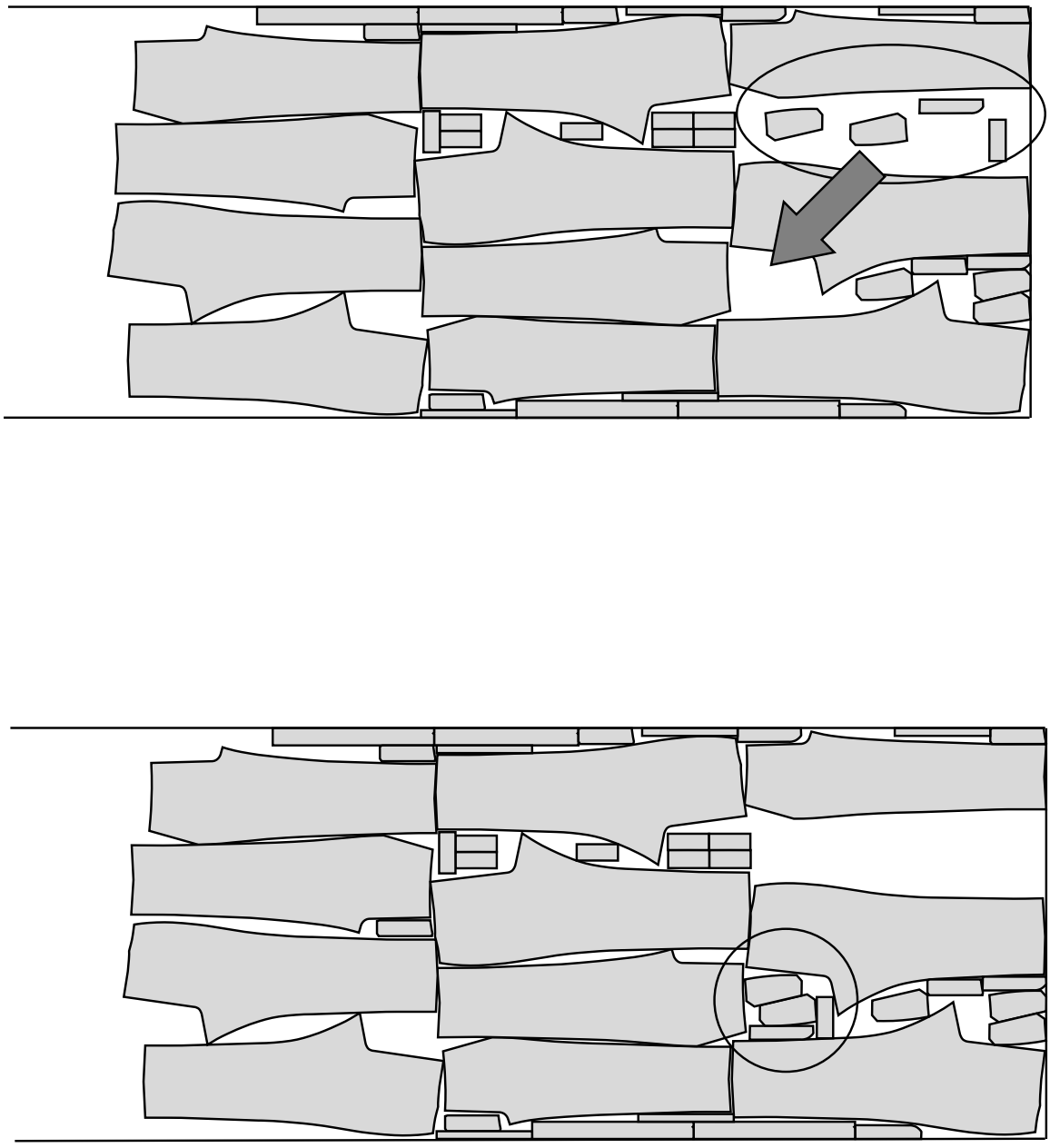


Figure 8.8: Example of trim placement using multiple containment algorithm.

Chapter 9

The Complexity of the Compaction Problem

9.1 Introduction

We define the *compaction* problem as a coordinated motion planning problem for a set of objects such that the area of the bounding convex hull or the bounding rectangle is minimized.

Since we are only interested in establishing lower bounds, we concentrate on a special case of compaction: all the objects are confined to move within a bounding rectangle whose top, left and bottom sides are fixed and whose right side can only move inward. It is easy to show that we can reduce this special case to the more general case of compaction by introducing a properly shaped sleeve piece and a piston piece. Therefore the lower bound of the special case is also a lower bound of the general case.

The coordinated motion planning problem for a collection of rectangles inside a fixed rectangular box (the warehouseman problem) has been studied by Hopcroft, *et. al.* [HSS84]. They obtained a well-known PSPACE-hardness result for the problem by a reduction from the PSPACE-hard symbol transposition problem. Spirakis and Yap [SY84] studied the coordinated motion of a collection of discs in a polygonal container and proved the strong NP-hardness of the problem. Since the possibility of PSPACE=P and NP=P has not been ruled out, the hardness results do not imply that it requires exponential time to decide the *existence* of a coordinated motion plan, even though the shortest plan might have an exponential number of moves. Chazelle *et. al.* [COSSW84] showed that a special case of

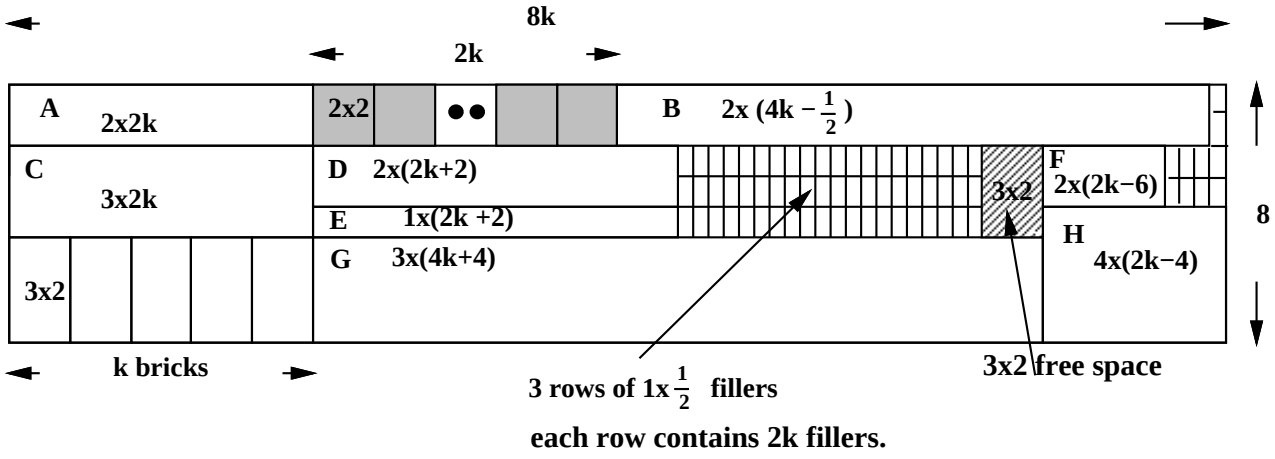


Figure 9.1: The reduction of warehouseman problem from the symbol transposition problem.

motion planning requires exponential number of moves. But in their case, only one object is allowed to move at a time. There is no previously known explicit construction of a coordinated motion planning that requires an exponential number of moves.

This chapter is organized as follows. In Section 9.2, we establish the PSPACE-hardness of the compaction problem, and we prove the existence of sets of rectangles which require an exponential number of moves to compact. In Section 9.3, we prove that if some simple non-rectangular objects are allowed, one can explicitly construct warehouseman and compaction problems which require a number of moves exponential in the number of edges in the input. In the last section, we show that if more complicated shapes are allowed, even finding a locally optimal solution to compaction might require an exponential number of moves. This result establishes that the position-based compaction algorithm presented in Chapter 4 has an exponential-time worst case.

9.2 The PSPACE-Hardness of Compaction

Hopcroft *et. al.* [HSS84] proved the PSPACE-hardness of the warehouseman problem by an ingenious reduction from the PSPACE-hard symbol transposition problem to the warehouseman problem. Here we follow their approach and reduce the transposition problem to the compaction problem. Our reduction is built on top of their construction with some additional construction to force the warehouseman problem to reach the final state when compaction has a solution. In the next section, we review the reduction used in [HSS84],

with emphasis on the facts that are related to our modification.

9.2.1 Review of the Warehouseman Problem

The layout of rectangular objects in the reduction in [HSS84] is shown in Figure 9.1. The bounding rectangle has dimension $8 \times 8k$ where k will be specified later (we follow the convention in [HSS84] to represent dimension as height $\times$ length). The large blocks A through H are called *sliders*. Their dimensions are shown in the figure. The $2 \times 2k$ space between A and B is filled with 2×2 blocks called *dominoes*, which play the role of symbols in the symbol transposition problem. The 3×2 blocks in the lower left corner are called *bricks*. The small $1 \times \frac{1}{2}$ blocks between D, E and F are called *fillers*. There are some additional fillers between B, F and the right boundary. The 3×2 free space to the left of F is the only free space within the bounding box.

There are two meaningful motions allowed in this layout. First, C, D, E and the fillers to right of D, E can move to the right to fill the 3×2 empty space. One brick below C can move up and fill the space opened by C. G can subsequently move to the left and 12 fillers can drop down to fill the space left by G. Such a circular motion can continue until all the bricks move up to the middle row, or can be reversed to move in the opposite direction. Second, during any step of the circular motion, free space can be opened up between C and D to allow a domino to drop down. Then B can move to the left, the fillers to the right of F can move up, and F can move next to the right wall. Eight fillers to the right of D and E can take the space to the left of F. A new 3×2 free space will be opened up to resume the circular motion, which can shift the domino between C and D to the left or right and reinsert it back into the row of dominoes. This latter motion simulates the transposition of symbols.

To simulate the transformation rules in the symbol transposition problem, each domino is further partitioned into smaller rectangles as shown in Figure 9.2. The partition of a domino consists of a bottom rectangle, a thick vertical spine, and a left and right set of rectangles. The spines all have the same dimension. The rectangles in the left and right set have decreasing heights when counted from bottom to top. The lengths of the rectangles are varied slightly around a nominal length to form various "dents" and "tabs" which can be used to enforce symbol transformation rules that govern whether a pair of dominoes can be neighbors.

In addition, there are objects called *spacers* between the dominoes. A spacer is much

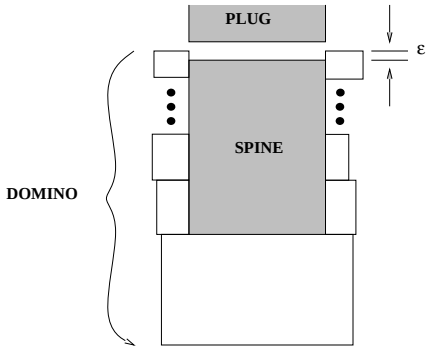


Figure 9.2: The construction of a domino.

longer than a domino so that it cannot be dropped down into the free space opened between C and D during the sequence of motion. A spacer can "transmit" the fit between a pair of dominoes on its two sides. The choices of the height of the rectangles in a domino together with the existence of the spacers ensure that (1) any pair of dominoes cannot be directly adjacent to each other; there must be at least one spacer between them (2) the rectangles of a domino cannot be rearranged within a domino and (3) the rectangles cannot be exchanged between two dominoes.

The amount k unspecified at the beginning can now be determined as the total length of all the symbols between block A and B (divided by 2).

Based on this construction, Hopcroft *et. al.* proved that the symbol transposition problem has a solution if and only if the dominoes in their initial configuration can reach a final configuration through a continuous coordinated motion.

9.2.2 PSPACE-hardness Proof

To reduce the symbol transposition problem to the compaction problem, we need to modify the construction so that the dominoes in their initial state, which represents an initial string in the transposition problem, can reach the final state if and only if the sliding wall in the compaction problem can move downward.

Let δ be the minimum height of the top rectangles in the left or right set of the dominoes. The height of the spine of a domino can be shortened by as much as $\epsilon = \frac{1}{5}\delta$ and still make the whole reduction work. This follows from the fact that the top rectangles in the left or right set of a domino have the smallest height among all the rectangles comprising a domino. Thus, the tiny free space on top of a spine cannot affect the two significant motions in the warehouseman problem, nor can it introduce new significant motions.

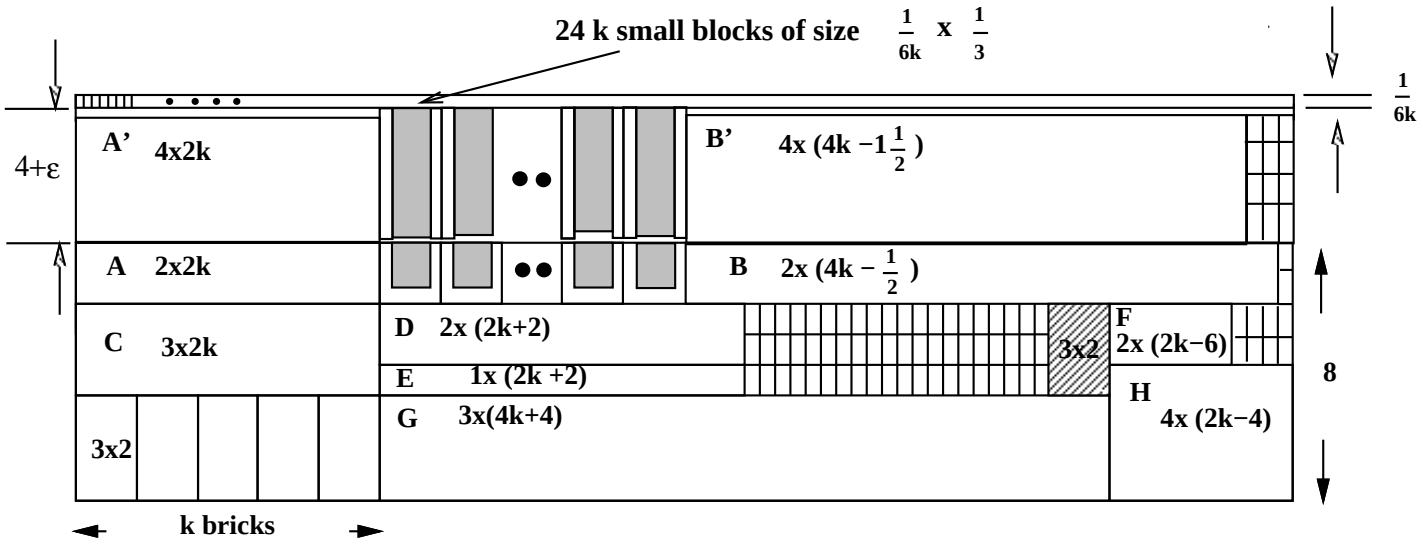


Figure 9.3: The reduction of compaction from the symbol transposition problem through the reduction of the warehouseman problem.

Assume there are n different types of dominoes. We shorten the height of the spine of a type i domino by $i\epsilon/n$. We introduce n types of *plug* blocks whose length equals that of the spine. A type i plug block has height $4+i\epsilon/n$. Thus, when a type i plug is put immediately on top of the spine of a domino, the total height is 6 units.

Figure 9.3 shows the additional constructions we need for the reduction. Let R denote the bounding box of the original construction. We will first delete the top edge of R and extend the two side edges upwards. We arrange the plugs in such a way that a plug of type i is put above where a type i domino would be in the final configuration of the warehouseman problem. Between the plugs, there are *stuffer* blocks of height 4 and length equal to the nominal length of the rectangles in a left or right set. Once plugs and stuffers are put into place, it is not possible for them to exchange positions. Initially, the top edges of the plugs and stuffers are aligned along a horizontal line which is $4+\epsilon$ above the top edge of block A. The maximum height of free space between the plug row and the domino row is at most 2ϵ (e.g. when a type 1 plug is put on top of a type n domino), which is less than half of the smallest height among dominoes rectangles. Once again, the existence of these tiny free spaces will not change the two significant motions inside R.

In order to force the dominoes to the desired final state, we must also take away the 3×2 free space inside rectangle R when the desired final configuration is reached. Otherwise, we can push D to the right by 2 units and drop a type 1 domino to the free space left between

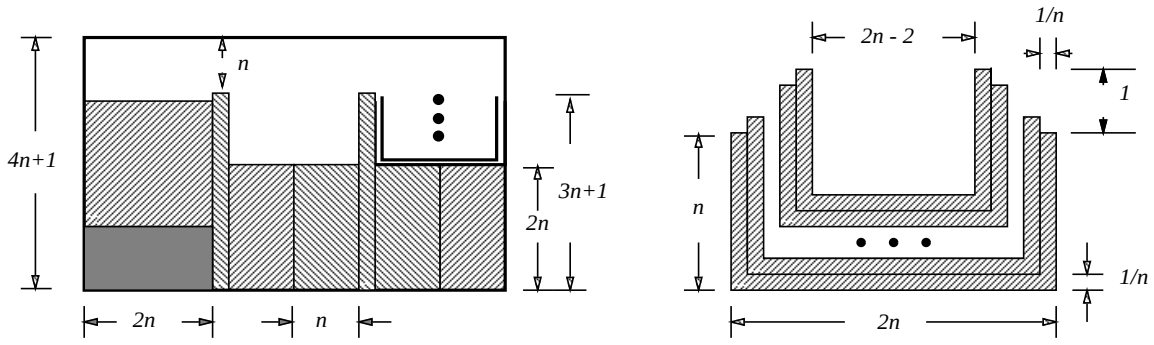


Figure 9.4: Construction and the initial state of a block puzzle.

C and D. If that happens, there can be many possible arrangements for the rest of the dominoes such that the sliding wall can drop down by ϵ. The 12 additional $1 \times \frac{1}{2}$ *fillers* to the right of B' can serve the purpose of filling up the free space. We need an extra layer of $24k$ small blocks of size $\frac{1}{6k} \times \frac{1}{3}$ to fill the space left open by the 12 fillers to the right of B'. If we put the sliding wall horizontally at the position $4 + \frac{1}{6k} + \epsilon$ above the top edge of block A, then the sliding wall can drop down by $\frac{1}{6k} + \epsilon$ if and only if the dominoes are in their final configuration. It is easy to see that this reduction, like the reduction in [HSS84], takes polynomial time. Thus we have

Theorem 9.1 *Compaction is PSPACE-hard even for a collection of rectangles.*

9.3 Compaction in an Exponential Number of Moves

The PSPACE-hardness of the symbol transposition problem shows that it is possible to simulate a binary counter by moving rectangular objects inside a rectangular container such that reaching a goal state from an initial state requires an exponential number of moves. But such a simulation may be complicated to construct and to describe. Here we show that if we relax the restriction that all the objects must be rectangles, we can have an example which is small in size and easy to understand. The example shows that even when the compaction problem involves simple objects such as U-shaped objects, we still have to deal with an exponential number of moves.

Our construction is shown in Figure 9.4. We build a rectangular container of dimension $(6n + 2) \times (4n + 2)$. Two vertical bars of width 1 and height $3n + 1$ divide the space within the container into three equal sized slots. We call them slot 1, 2 and 3, from left to right. The bottom half of each slot is taken by some rectangular blocks. There is one $2n \times 2n$

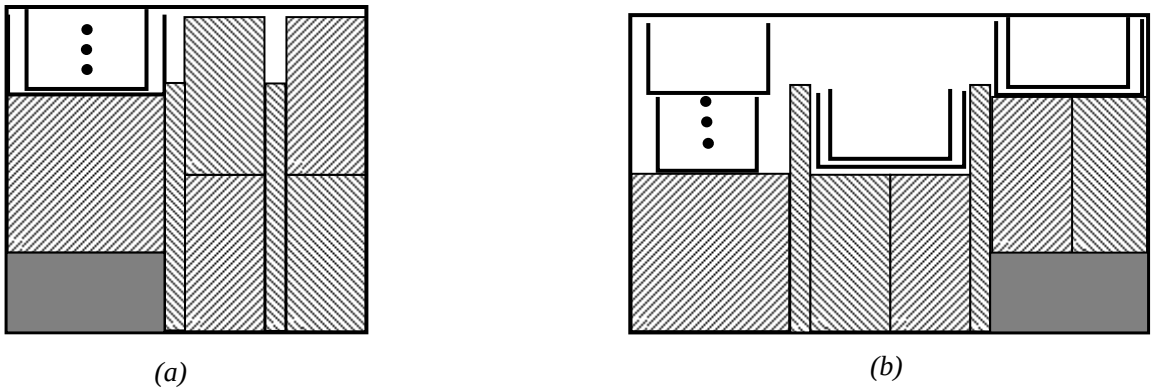


Figure 9.5: (a) The final state. (b) A dead-end situation.

block in slot 1 and two $n \times 2n$ blocks on each of slots 2 and 3. Due to the vertical bars, these rectangular blocks cannot be moved from one slot to another. On the upper half of the third slot, there are some U-shaped objects: $U_1, U_2, \ldots, U_n$ stacked together vertically (U_n is the smallest one). The walls of these objects have width $\frac{1}{n}$. All of these objects have height n. The bottom of wall of U_i is $\frac{2n-2i}{n}$ long.

The height of the vertical bars ensures that only one U-shaped object can pass through the openings on top of the bars. Therefore, at any particular move, only the object on top of a stack can be moved. There is a $2n \times n$ block initially at the bottom of the left slot. This block can slide between the bottom of the slots and is called a *slider*.

It is easy to see that the right wall of the container can be moved to position $4n + 2$ if and only if all the U-shaped objects are moved to the upper half of the first slot (see Figure 9.5 (a)).

It is easy to verify that to solve the compaction problem, a problem harder than the $n - 3$ disc Tower of Hanoi problem has to be solved[1]. Thus the total motion required is at least $\Omega(2^{n-3})$, which is exponential in n.

Theorem 9.2 *Coordinated motion planning requires a number of moves that is exponential in the number of edges in the input.*

Proof: Omitted. □

[1]Discs are restricted to move to neighboring posts. This is still solvable with $O(3^n)$ moves.

9.4 Finding a Local Minimum Requires an Exponential Number of Moves

In this section, we first give another construction that involves more complicated shapes to show that compaction requires an exponential sized motion. Then, based on the construction, we show that even finding a local optimum, according to an algorithm presented in Chapter 4, requires an exponential number of steps.

Consider the device shown in Figure 9.6. The device is composed of four movable pieces and a static obstacle that also provides the top, left and bottom boundaries. The obstacle is a single connected piece and has a row of k "teeth" on the top and a row of k "teeth" on the bottom. A tooth is one unit wide and one unit high. The teeth on the top are displaced a half unit to the right with respect to the teeth on the bottom. The four movable pieces are labeled **A, B, C** and **D** as shown in Figure 9.6. The goal is to move piece A to the left by k units. To this end, we need to move piece **B** through the two rows of teeth. During the motion, a total of k up and down motions will occur for piece **B**. **B** moves one unit vertically during each up or down motion. One unit of vertical motion of **B** will be transferred through piece **C** to piece **D**. The two slanted edges in piece **D** have slope $\frac{1}{k}$. Therefore, each up (down) motion of one unit in **B** will force **D** to move to the right (left) by k units.

The construction is actually replicated n times and concatenated together as shown in Figure9.7. The top part is connected to the bottom part ultimately at the far left. The pieces are organized into n groups with each group containing four pieces. There are overlaps among components in the groups since the piece D in one group also serves as piece A in the next group. From left to right, the groups are ordered from 1 to n. By the analysis in the previous paragraph, the k units of horizontal motion of piece **A** of group 1 force k^2 units of horizontal motion of piece **D** of group 1, which is the piece **A** of group 2. The k^2 units of horizontal motion of piece **A** of group 2 in turn force k^3 units of horizontal motion of piece **A** of group 3. When the motion propagates to group n, piece **A** of group n has to move a total of k^n units horizontally. If we choose k as a small constant, then the total number of edges in the construction is linear in n.

Hence, we have the following theorem.

Theorem 9.3 *The compaction problem shown in Figure 9.7 requires an exponential number of moves.*

The motivation for the construction in Figure 9.7 is to show that the compaction algorithm presented in Chapter 4 for finding a local minimum of the compaction problem needs an exponential number of iterations in the worst case. To make the construction applicable to our problem, we must show that the description of the motion takes only polynomial space. To show this, we observe that piece **B** of group n is the busiest piece. We divide the motion into basic steps according to the motion of the piece. The motion between the moment that piece **B** of group n moves from the highest vertical position to the lowest one (or vise versa) is called a *basic step*. At each basic step, only piece **B** in group n can move a full vertical and horizontal unit. The distance that can be moved by a piece in another group declines exponentially with group number. For example, the piece **B** in group 1 can only move $O(k^{-n})$ horizontal or vertical units in each basic step. This shows that at each basic step, a description of the state of the system requires only polynomial space. Hence, polynomial space is sufficient to describe the complete motion which is composed of basic steps.

Recall that the compaction algorithm in Chapter 4 uses a configuration space approach for finding a local optimum. For each neighboring pair of objects (represented as polygons), the Minkowski sum of the two objects is generated. A large convex subset of the exterior of the Minkowski sum is determined according to a locality heuristic. A set of linear inequalities are built using the boundary edges of the convex subset. The relative motion of the pair of polygons is constrained to stay inside the convex subset by the set of linear inequalities. A linear program is set up using the set of linear inequalities for every neighboring pair as constraints. The objective function maximizes the motion of an imaginary "piston" piece at the right boundary. The solution of the linear program will give the new positions for all the objects. A new set of inequalities is set up if the convex subsets outside the Minkowski sum of a pair of objects changes. The algorithm iterates until there is no improvement to the objective function.

If we apply the algorithm to the compaction problem shown in Figure 9.7, it is easy to show that each iteration of the algorithm finds exactly the configuration after each basic step. The objective function can be improved until piece A of group 1 is moved to the left by k units. Because there is a total of $\Theta(k^n)$ basic steps, and each iteration of the algorithm corresponds to exactly one basic step, there will be a total of $\Theta(k^n)$ calls to the linear program.

To show that the algorithm can only find a local minimum, we make two dents in piece

A of group 1. The two dents have the same height h but different widths d_1 and d_2. Assume that the lower dent has width d_1 and $d_1 < d_2$. We make a rectangular object with dimension $d_2 \times h$. The rectangular object is originally placed at the lower dent. According to the heuristics used in the algorithm, the algorithm will never move the rectangular object from the lower dent to the higher dent. Hence, the algorithm does not find a global optimum when the algorithm terminates.

Theorem 9.4 *The compaction algorithm described in Chapter 4 requires an exponential number of iterations to find a local minimum.*

We note that the example shown in Figure 9.7 demonstrates the worst case behavior of the compaction algorithm presented in Chapter 4. In practice, the algorithm runs very fast. For the application of compacting garment layouts consisting of up to 120 pieces, the algorithm always terminates in less than 10 iterations.

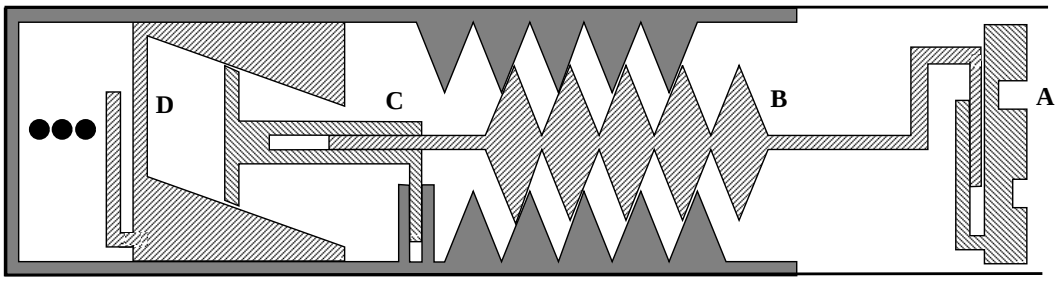


Figure 9.6: The construction of a motion propagation device.

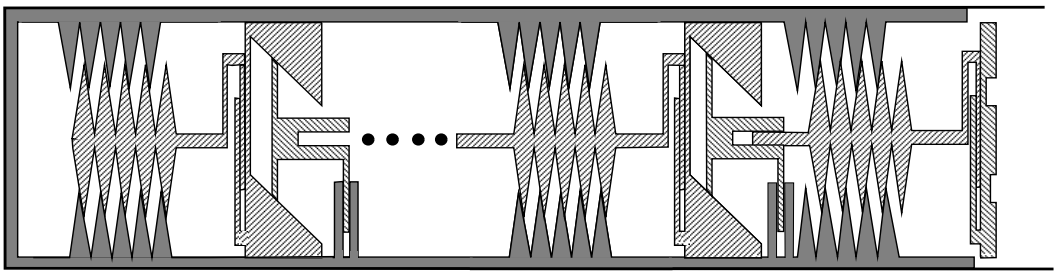


Figure 9.7: The replication and concatenation of the motion propagation device.

Chapter 10

Conclusion

The position-based compaction algorithm presented in this thesis is the first fast and practical algorithm for compacting a layout of convex and non-convex polygons. The algorithm has been shown to be very effective in a real world application: marker making in apparel manufacturing. The CAD company who licensed our software has been conducting on site experiments on the markers of some of their client companies. Preliminary results indicate that the software could save each client between a quarter of a million to a million dollars per year. Potential saving to the entire textile industry amount to tens of millions of dollars per year.

The algorithm is general and versatile. We can use the algorithm to perform different tasks in all stages of two-dimensional packing process, such as compaction, opening up gaps and database driven automatic marker making, by simply tailoring the objective functions. We have also shown that the algorithm can be extended to handle small degree rotations, reduce the total amount of overlaps in a layout/perform floating and find optimal two-dimensional layout of polygons.

The key to the position-based compaction algorithm is the idea of representing the non-convex free space of motion planning locally by a large convex subset identified by a locality heuristic. The convex subsets make it possible to directly calculate the position of the polygons by building and solving a linear program model. Even though in the worst case compaction takes exponential number of moves of the polygons, in practice, the algorithm always converges and find a local optimum that improves the existing layout in a few iterations.

During its execution, the algorithm always keeps the layout in a non-overlapping configuration. This characteristic is important in compacting tightly packed layout because

if not planned carefully, an overlap in the layout, once occurred, can be hard to eliminate due to the tightness of the layout. The requirement of avoiding introducing overlaps makes techniques such as simulated annealing less applicable to generating or compacting tightly packed layouts since the some basic steps of simulated annealing such as swapping two polygons or perturb the current position or orientation (flipping and rotation) of a polygon are very likely to introduce overlaps.

The work presented in this thesis lies between the boundary of computational geometry and operations research. The algorithms developed here combine the concepts and techniques from both fields. One reason of the effectiveness of our algorithms is that they utilize the power of the mature linear programming techniques implemented in a state of the art linear programming package. The linear program package performs the basic steps of our algorithms fast and reliably. As exemplified in the position-based compaction algorithm, the linear programming formulation can greatly improve the running time over the previous non linear programing formulations.

The numerical instability of geometric computations is a major problem in implementing geometric algorithms [Mil88]. An additional advantage of using a linear programming package is to use its numerical stability to overcome the numerical difficulties of geometric computations. Combined with robust computational geometry techniques, such as the robust computation of Minkowski sums, our position-based compaction algorithm proves to be quite numerically stable in practice.

Perhaps the most important impact of our position-based compaction algorithm on two-dimensional packing/polygon nesting problems arises from its ability to separate overlapping polygons. This ability has great impact because it leads to a database driven automated marker making scheme. This scheme provides a very viable approach for totally automating layout generation process, which is our ultimate goal. The limitation of automatically generating layouts using geometric algorithms alone is that in order to achieve efficiencies comparable to that of human generated layouts, one has to develop special techniques for different application domains. For example, efficient techniques for making jacket markers might be quite different from those of making pants markers. The database driven marker making scheme, on the other hand, can be thought as an implicit form of learning from examples. The human marker makers' domain knowledge, their many years of experience, and especially their understanding of the relationship among various shapes are "condensed" into a set of examples – the collection of high quality markers in the database. With sufficient

examples in the database for the matching algorithm to find a close match, the database drive marker making scheme can produce a layout that is close to human performance.

We conclude this thesis with a list of future research directions. First, we list the problems related to extending the position-based compaction algorithm and improving its performance.

Improving the Rotational Compaction Algorithm It is beneficial to better understand the discrepancies in compaction results produced by the two rotational compaction algorithm. We hope that our understanding can leads to improved rotational compaction algorithms.

Compaction of Large Layouts Right now, the position-based compaction algorithm can compact a layout of one hundred polygon in less than one minute (mostly between 10 to 20 seconds). For several large examples, we have observation long running time. This might be caused by the large number of constraints. We are interested in improving the running time. There are two reasons for study this. First, there exist large layouts in practice. Second, we have observed that the material utilization can further be improved by concatenating several markers into a single large one.

Second, we list problems related to database driven marker making.

Shape Matching Algorithms The key to producing a high quality layout in the database driven marker making scheme is to find a close match between the polygons in the layout to be made and those in an layout in the database. Therefore, improving the shape matching algorithm directly improves the quality of a automatically generated layout.

Improved Overlap Elimination Algorithms From Chapter 5, we see that if there are no limitations on the width and the length of a layout, then the position-based algorithm can always find a separation of the overlapped polygons. However, in strip packing applications, the width of the layout is fixed. Therefore, the separation algorithm can fail to eliminate all the overlaps. To overcome this difficulty, we can use techniques from linear programming to analyze the slack variables associated with the constraints. From these slack variables, we then find the constraints that are most "tight" and further identify the polygons involved. We can then "pop out" these polygons from the layout and eliminate the overlaps among the remaining polygons.

Next, we can shrink the polygons by certain amount and find the gaps that can accommodate the shrunken polygons. To put the polygons back into the layout, we can perform the following steps: (1) place the shrunken polygon, (2) restore it to its original size and (3) eliminate the resulting overlaps using our algorithm. If overlaps still cannot be eliminated, it would be necessary to repeat with a different polygon. Based on this idea, we can design more sophisticated algorithms for separating polygons.

Organization of the Database In database driven marker making, if the database contains tens of thousands of markers, it becomes crucial to organize the database such that a marker can be compared with and retrieved quickly. Despite some recent interests [Jag91], the area is wide open and offers great challenges as well as opportunities.

Appendix A

Vectors and Cross Products

Let A, B and C be three points in the plane with coordinates are (A_x, A_y), (B_x, B_y) and (C_x, C_y) respectively in a coordinate system whose origin is at point O. We denote by $\mathbf{A}$ the vector originated from O and ended at A. Vector $\mathbf{A}$ can be expressed as a linear combination of $\mathbf{i}$ and $\mathbf{j}$, the unit vectors in the x and y axis, as:

$$\mathbf{A} = A_x\,\mathbf{i} + A_y\,\mathbf{j}$$

We denote by $\mathbf{AB}$ the vector originated from A and ended at B. This vector represents the relative displacement of point A with respect to point B. From Figure A.1, we can easily see

$$\mathbf{AB} = \mathbf{B} - \mathbf{A} = (B_x - A_x)\,\mathbf{i} + (B_y - A_y)\,\mathbf{j}$$

Let $AB_x = B_x - A_x$ and $AB_y = B_y - A_y$, $\mathbf{AB}$ can be written as:

$$\mathbf{AB} = AB_x\,\mathbf{i} + AB_y\,\mathbf{j}$$

By standard definition, the cross product of vectors $\mathbf{AB}$ and $\mathbf{BC}$ is given by

$$\mathbf{AC} \times \mathbf{BC} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ AC_x & AC_y & AC_z \\ BC_x & BC_y & BC_z \end{vmatrix}$$

where AC_z and BC_z are the z components of AC and BC which are both zero in our case. Expanding the determinant, we have

$$\begin{aligned} \mathbf{AC} \times \mathbf{BC} &= [(AC_x \cdot BC_y - AC_y \cdot BC_x)]\,\mathbf{k} \\ &= [(C_x - A_x) \cdot (C_y - B_y) - (C_y - A_y) \cdot (C_x - B_x)]\,\mathbf{k} \end{aligned}$$

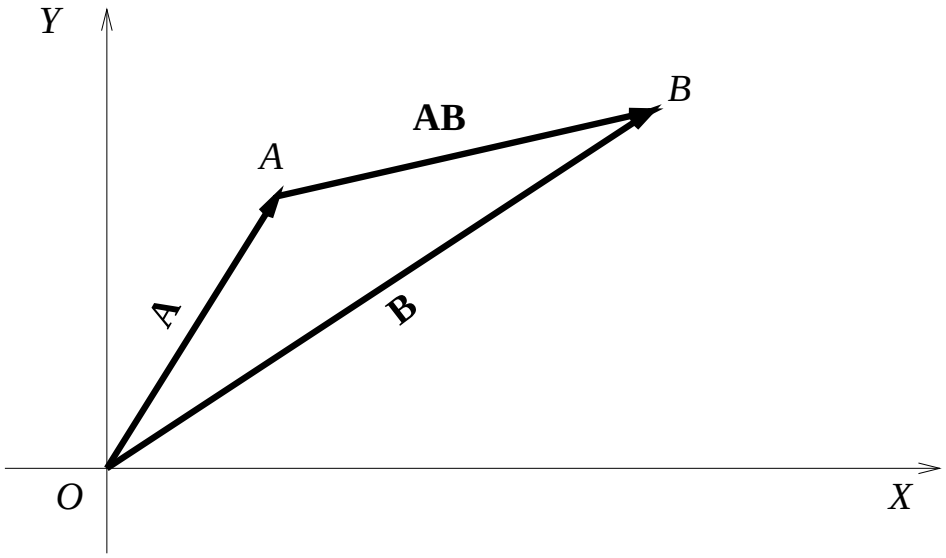


Figure A.1: Definition of vectors

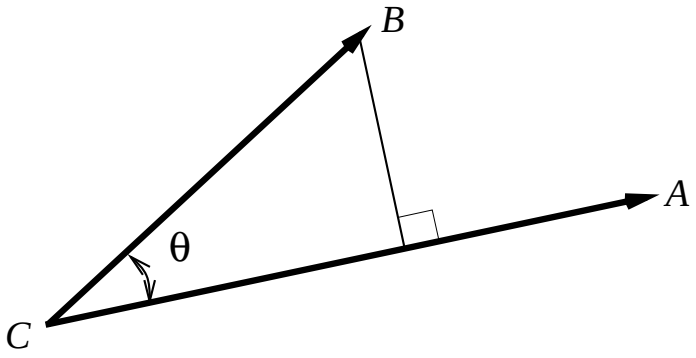


Figure A.2: The geometric interpretation of cross product

Therefore, the cross product of two planar vectors is a vector parallel to the z-axis. The direction of the cross product is the same as the positive z axis if and only if **AC** and **BC** form a right hand system. The geometric interpretation of the $\mathbf{AC} \times \mathbf{BC}$ is that its value is twice the signed area of the triangle $\triangle(A, B, C)$ and the sign is positive if and only if A, B and C are in clockwise order (see Figure A.2).

Since in our applications we are only interested in the scalar part of the cross product, we define the cross product of two planar vectors as a scalar directly.

Definition A.1 *Given three points A, B and C in the plane, the cross product of vectors*

AC *and* **BC** *is defined as:*

$$\mathbf{AC} \times \mathbf{BC} = (C_x - A_x) \cdot (C_y - B_y) - (C_y - A_y) \cdot (C_x - B_x)$$

The following fact follows immediately from the geometric interpretation of the standard defintion of cross product.

Fact A.1

$$\mathbf{AC} \times \mathbf{BC} = |AC||BC| \sin\theta$$

where θ *is the angle traversed when we sweep* **AC** *counterclockwisely until it reaches* **BC**.

By rewriting the right hand side of our definition of cross product into a determinant form, we obtain the following fact.

Fact A.2

$$\mathbf{AC} \times \mathbf{BC} = \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix}$$

It is shown in [GS85] that the well known CCW ($CounterClockWise$) predicate[1] for three point A, B and C can be expressed by the condition

$$\begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} > 0$$

Fact A.2 shows that the CCW predicate is equivalent to $\mathbf{AC} \times \mathbf{BC} > 0$.

Combining the above two facts, we obtain the following lemma which is extremely useful in deriving non-penetration constraints.

Lemma A.3 *Let* A, B *and* C *be three points in the plane,*

$$A, B \text{ and } C \text{ are } \left\{ \begin{array}{c} \text{in counterclockwise order} \\ \text{co-linear} \\ \text{in clockwise order} \end{array} \right\} \text{iff } \mathbf{AC} \times \mathbf{BC} \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} 0$$

For notational simplicity, we sometimes also use $AC \times BC$ to denote the cross product $\mathbf{AC} \times \mathbf{BC}$ in this thesis.

[1] For three points $A(A_x, A_y)$, $B(B_x, B_y)$ and $C(C_x, C_y)$ in the plane, the predicate $CCW(A, B, C)$ is true if and only if A, B and C are in counterclockwise order.

Bibliography

[ABB91] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. In *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, pages 186–193, 1991.

[AG92] H. Alt and M. Godau. Measuring the resemblance of polygonal curves. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 102–109, 1992.

[AKS90] R. Anderson, S. Kahan, and M. Schlag. An $o(n \log n)$ algorithm for 1-d tile compaction. In M. Nagl, editor, *Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 287–301. Springer-Verlag, 1990.

[AST92] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 72–82, 1992.

[Bar89] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (Proc. SIGGRAPH)*, 23(3):223–232, 1989.

[BB88] R. Barzel and A. H. Barr. A modeling system based on dynamics constraints. *Computer Graphics (Proc. SIGGRAPH)*, 22(4):179–187, 1988.

[BECR80] Brenda S. Baker, Jr. E.G. Coffman, and Ronald L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 9(4):846–855, 1980.

[Can87] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1987.

[CE88] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 590–600, 1988.

[Cha90] B. Chazelle. Triangulating a simple polygon in linear time. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 220–230, 1990.

[CKSS81] Seth Chaiken, Daniel J. Kleitman, Michael Saks, and James Shearer. Covering regions by rectangles. *Siam. J. Alg. Disc. Meth.*, 2(4):394–410, 1981.

[CMD82] F.R.K. Chung, M.R.Garey, and D.S.Johnson. On packing two-dimensional bins. *SIAM Journal of Alg. Disc. Meth.*, 3(1):66–76, 1982.

[COSSW84] B. Chazelle, T. Ottmann, E. Soisalon-Soininen, and D. Wood. The complexity and decidability of SEPARATION^{TM}. In *Proc. 11th Internat. Colloq. Automata Lang. Program.*, volume 172 of *Lecture Notes in Computer Science*, pages 119–127. Springer-Verlag, 1984.

[DD92] Kathryn A. Dowsland and William B. Dowsland. Packing problems. *European Journal of Operational Research*, 56:2 – 14, 1992.

[DLM93] K. Daniels, Z. Li, and V. Milenkovic. Multiple containment methods and applications in cloth manufacture. In Joseph Mitchell, editor, *Proceedings of the Army Research Office and MSI Stony Brook Workshop on Computational Geometry*. SUNY Stony Brook, October 1993.

[DLM94] K. Daniels, Z. Li, and V. Milenkovic. Multiple containment methods. Technical Report TR-12-94, Center of Research in Computing Technology, Aiken Computation Laboratory, Harvard University, 1994.

[DM94] K. Daniels and V. Milenkovic. Limited Gaps. In Mark Keil, editor, Submitted to *the Sixth Canadian Conference on Computational Geometry*, Saskatoon, Canada S7N 0W0, keil@cs.usask.ca, August 1994. University of Saskatchewan.

[Dyc89] Haralk Dyckhoff. A typology of cutting and packing problems. *European Journal of Operations Research*, 44:145–159, 1989.

[ECL91] Jr. E.G. Coffman and George S. Lueker. *Probabilistic Analysis of Packing and Partitioning Algorithms*. Wiley and Sons, Inc., New York, 1991.

[ECS90] Jr. E.G. Coffman and P.W. Shor. Average-case analysis of cutting and packing in two dimensions. *European Journal of Operations Research*, 44:134–144, 1990.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, West Germany, 1987.

[G$\ddot{7}$9] F. Göbel. Geometrical packing and covering problems. In A. Schrijver, editor, *Packing and Covering in Combinatorics*, pages 179–199. Mathematical Centre Tracts, 1979.

[Gar79] Martin Gardner. Mathmatical games: some packing problems that cannot be solved by sitting on the suitcase. *Scientific American*, 241(4):18–26, 1979.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[GG63] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem - Part II). *Operations Research*, 11:863–888, 1963.

[GJ79] Mickael R. Garey and Davis S. Johnshon. *Computers and Intractability – A guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[GJPj78] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tar jan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7:175–179, 1978.

[GRS83] L. Guibas, L. Ramshaw, and J. Stolfi. A Kinetic Framework for Computational Geometry. In *IEEE 24th Annual Symposium on Foundations of Computer Science*, 1983.

[GS85] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[Her89] J. Hershberger. Finding the upper envelope of n line segments in $o(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[HM83] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 1983.

[HS86] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

[HSS84] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: $PSPACE$-hardness of the "Warehouseman's Problem". *Internat. J. Robot. Res.*, 3(4):76–88, 1984.

[Jag91] H. V. Jagadish. A retrieval technique for similar shapes. In *Proc. ACM SIGMOD Conf. on the Management of Data.*, pages 208–217, 1991.

[Kei85] J. M. Keil. Decomposing a polygon into simpler components. *SIAM J. Comput.*, 14:799–817, 1985.

[KOS91] A. Kaul, M.A. O'Connor, and V. Srinivasan. Computing Minkowski Sums of Regular Polygons. In *Proceedings of the Third Canadian Conference on Computational Geometry*, Vancouver, British Columbia, 1991.

[Len84] T. Lengauer. On the solution of inequality systems relevant to ic-layout. *Journal of Algorithms*, 5(5):408–421, 1984.

[LM90] Z. Li and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 235–243, 1990.

[LM92] Z. Li and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. *Algorithmica*, 8:345–364, 1992.

[LM93a] Z. Li and V. Milenkovic. A compaction algorithm for non-convex polygons and its application. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 153–162, 1993.

[LM93b] Z. Li and V. Milenkovic. Compaction of a 2D layout of non-convex shapes and applications. In *SIAM Conference on Geometric Design*. SIAM, November 1993.

[LM93c] Z. Li and V. Milenkovic. On the complexity of the compaction problem. In *Proc. 5th Canadian Conference on Computational Geometry*, pages 153–162, 1993.

[LMar] Z. Li and V. Milenkovic. Compaction and seperation algorithms for non-convex polygons and their applications. *European Journal of Operations Research (Special Issue on Cutting and Packing)*, To appear.

[LPW79] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22:560–570, 1979.

[LZ] Z. Li and B. Zhu. On the monotonicity of polygons and polyhedra. To be submitted.

[Mal90] F. Miller Maley. *Single-layer Wire routing and compaction.* The MIT Press, 1990.

[Map90] D. Maple. A hierarchy preserving hierarchical compaction. In *Proc. 27th Design Automation Conference*, pages 375–381, 1990.

[MDL91] V. Milenkovic, K. Daniels, and Z. Li. Automatic Marker Making. In *Proc. 3rd Canadian Conference on Computational Geometry*, 1991.

[MDL92] V. Milenkovic, K. Daniels, and Z. Li. Placement and Compaction of Nonconvex Polygons for Clothing Manufacture. In *Proc. 4th Canadian Conference on Computational Geometry*, 1992.

[Meg84] N. Megiddo. Linear programming in linear time when the dimension is fixed. *JACM*, 31:114–127, 1984.

[MFS87] R. C. Mosteller, A. H. Frey, and R. Suaya. 2-d compaction a monte carlo method. In P. Lasleber, editor, *Advanced Research in VLSI*, pages 173–197. MIT Press, 1987.

[Mil88] V. Milenkovic. *Verifiable implementation of geometric algorithms using finite Precision Arithmetic.* PhD thesis, Carnegie Mellon University, 1988. CMU-CS-88-168.

[Mur87] Frank D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM Journal of Computing*, 16(1):149–161, 1987.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Computer Graphics (Proc. SIGGRAPH)*, 22(4):289–298, 1988.

[Nem93] G. Nemhauser. The age of optimization – solving large scale real world problems. Philip McCord Morse Lecture, ORSA/TIMS Annual Meeting, Chicago, May 1993.

[PB88] J. C. Platt and A. H. Barr. Constraint methods for flexible models. *Computer Graphics (Proc. SIGGRAPH)*, 22(4):279–287, 1988.

[PS81] F.P. Preparata and K. Supowit. Testing a simple polygon for monotonicity. *Inform. Process. Lett.*, 12(4):161–164, 1981.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.

[Rei79] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 421–427, 1979.

[Rei87] J. H. Reif. Complexity of the generalized movers problem. In J. Hopcroft, J. Schwartz, and M. Sharir, editors, *Planning, Geometry and Complexity of Robot Motion*, pages 267–281. Ablex Pub. Corp., Norwood, NJ, 1987.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.

[SP92a] P. E. Sweeney and E. R. Paternoster. Cutting and packing problems: A categorized, application-orientated research bibliography. *J. Opl Res. Soc.*, 43(7):691–706, 1992.

[SP92b] Paul E. Sweeney and Elizabeth Ridenour Paternoster. Cutting and packing problems: A categorized, application-oriented research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[SS90] J. T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pages 391–430. Elsevier, Amsterdam, 1990.

[SSVS86] H. Shin, A. L. Sangiovanni-Vincentelli, and C. H. Séquin. Two dimensional compaction by 'zone refining'. In *Proc. 23rd Design Automation Conference*, pages 115–122, 1986.

[SY84] P. Spirakis and C. Yap. Strong NP-hardness of moving many discs. *Inform. Process. Lett.*, 19:55–59, 1984.

[TV88] R. E. Tarjan and C. J. Van Wyk. An $O(n \log\log n)$-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988. Erratum in 17(1988), 106.

[Won85] C. K. Wong. An optimal two-dimensional compaction scheme. In P. Bertolazzi and F. Luccio, editors, *VLSI: Algorithms and Architectures*, pages 205–211. Elsevier (North-Holland), 1985.